MATLAB仿真与应用系列丛书

本书提供源代码下载

MATLAB 神经网络仿真与应用

张德丰 编著

MATLAB 2008



電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

http://www.phei.com.cn

MATLAB程序设计与典型应用

MATLAB/Simulink建模与仿真

MATLAB在电子信息工程中的应用

MATLAB控制系统设计与仿真

MATLAB神经网络仿真与应用

本书提倡的学习模式：

通过案例提出问题



介绍解决问题的方法



归纳总结，培养寻找答案的思维习惯





策划编辑：陈韦凯
责任编辑：张　帆

ISBN 978-7-121-08923-7



定价：39.00元

MATLAB 仿真与应用系列丛书

MATLAB 神经网络仿真与应用

张德丰　编著

電子工業出版社

Publishing House of Electronics Industry

北京 • BEIJING

内容简介

本书共分为 10 章。主要内容包括人工神经网络的分类、MATLAB 神经网络工具箱的对象与属性、神经网络工具箱函数的分析及实例、感知器、线性神经网络、BP 网络、径向基网络、GMDH 网络、自组织竞争型神经网络、自组织特征映射神经网络、自适应共振理论模型、对向传播网络、Elman 神经网络、Hopfield 网络、联想记忆、BSB 模型及其应用、图形用户接口、Simulink 仿真、自定义神经网络、神经网络在工程中的应用等内容。

本书可作为高等院校计算机、电子工程、控制工程、信息与通信科学、数学、机械工程和生物医学工程等专业学生的参考教材，对从事上述领域工作的广大科技人员具有重要的参考价值，对学习神经网络及其仿真技术的读者来说，也是一本极为有用的入门指导书。

图书在版编目（CIP）数据

MATLAB 神经网络仿真与应用 / 张德丰编著.—北京：电子工业出版社，2009.6
（MATLAB 仿真与应用系列丛书）

ISBN 978-7-121-08923-7

I. M… II.张… III.计算机辅助计算－软件包，Matlab－应用－神经网络－计算机仿真 IV.TP391.75 TP183

中国版本图书馆 CIP 数据核字（2009）第 082101 号

策划编辑：陈韦凯
责任编辑：张　帆
印　　刷：北京市天竺颖华印刷厂
装　　订：三河市鑫金马印装有限公司
出版发行：电子工业出版社
　　　　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×1092　1/16　印张：22　字数：592 千字
印　　次：2009 年 6 月第 1 次印刷
印　　数：4 000 册　定价：39.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

丛书编委会

丛书序言

MATLAB 一词是 Matrix Laboratory（矩阵实验室）的缩写。20 世纪 70 年代后期，时任美国新墨西哥大学计算机科学系主任的 Cleve Moler 教授为减轻学生编程负担，为学生设计了一组调用 LINPACK 和 EISPACK 库程序的“通俗易用”的接口，此即用 Fortran 编写的萌芽状态的 MATLAB。此后，MATLAB 软件的功能便不断得到丰富和发展。

在欧美大学里，诸如应用代数、数理统计、自动控制、数字信号处理、模拟与数字通信、时间序列分析、动态系统仿真等课程的教科书把 MATLAB 作为一项重要学习内容。这几乎成了 20 世纪 90 年代以后教科书与旧版书籍的区别性标志。

在国际学术界，MATLAB 已经被确认为准确、可靠的科学计算标准软件。MATLAB 将数值分析、矩阵运算、信号处理、图形功能和系统仿真融为一体，使用户在易学易用的环境中求解问题，如同书写数学公式一样，避免了传统的复杂专业编程。

本套丛书是编委会经过对多所高等院校和知名企业进行调研，在与各高校教师和数十位不同领域工程师广泛交流的基础上编写的。编委会成员都是来自计算机教学的一线教师和就职于各知名企业的工程师，具有非常丰富的教学和实践经验。

本套丛书是以 MATLAB R 2008 为平台来讲解各学科知识的，它也适合其他 MATLAB 版本，具有如下主要特点：

（1）突出技术，全面针对实际应用。在选材上，根据实际应用的需要，坚决舍弃现在用不上、将来也用不上的内容。在保证学科体系完整的基础上不过度强调理论的深度和难度，注重应用型人才的专业技术和工程实用技术的培养。此系列丛书从内容上讲，跨度较大，从 MATLAB 在基础层面的应用到专业工具箱的高层次的应用，这样不仅满足不同领域和不同层次读者的需要，还使读者可以根据自己的水平自主选用。

（2）本套丛书采用“任务驱动”的编写方式，采取“提出问题——介绍解决问题的方法——归纳总结，培养寻找答案的思维方法”的模式。以实际问题引导出相关的原理和概念，在讲述实例的过程中将知识点融入，通过分析归纳，介绍解决工程实际问题的思想和方法，最后进行概括总结，使书中内容层次清晰，脉络分明，可读性和操作性强。同时引入案例学习和启发式学习方法，便于激发学习兴趣。

（3）内容安排上力求由浅入深，循序渐进；表述清晰，通俗易懂；讲求效率，内容经过多次提炼和升华，突出规律和学习技巧，是思维化的直接体现。

（4）充分体现案例学习模式。在本系列丛书中读者会发现，凡是讲解一个问题都以一个案例为主线进行阐述，这是本系列丛书作者多年来在教学第一线的经验总结。案例学习引人入胜，易理解，易掌握，能使读者举一反三，技术掌握扎实。

我们力争使这套丛书在可读性、指导性和实用性上达到最优；但肯会有不尽人意之处，诚挚接受广大读者的批评、指正。同时也希望与读者在本套丛书的学习、应用上相互交流，来信可发往 zhangdf@foshan.net。

编 委 会

2009 年 3 月

前　言

MATLAB 是 MathWorks 公司推出的一套高性能的数值计算和可视化软件，它的推出得到了各个领域专家学者的广泛关注，其强大的扩展功能为用户提供了强有力的支持；它集数学运算、图形绘制、语言设计和神经网络等 30 多个工具箱于一体，具有极高的编程效率。

快速发展的 MATLAB 软件为神经网络理论的实现提供了一种便利的仿真手段。MATLAB 神经网络工具箱的出现，更加拓宽了神经网络的应用空间。神经网络工具箱将很多原本需要手动计算的工作交给计算机，一方面提高了工作效率；另一方面，还提高了计算的准确度和精度，减轻了工程人员的负担。

本书使用 MATLAB 神经网络工具箱作为开发平台，以神经网络理论为基础，利用 MATLAB 脚本语言构造出典型神经网络的激活函数，如线性、竞争性和饱和线性等激活函数，使设计者对所选定网络输出的计算变成对激活函数的调用。另外，根据各种典型的修正网络权值的规则，再加上网络的训练过程，利用 MATLAB 编写出各种网络设计和训练的子程序，网络设计人员可以根据自己的需要去调用工具箱中有关的设计和训练程序，将自己从烦琐的编程中解脱出来，集中精力解决其他问题，从而提高了工作效率。

最新版本的神经网络工具箱几乎涵盖了所有的神经网络的基本常用模型，如感知器和 BP 网络等。对于各种不同的网络模型，神经网络工具箱集成了多种学习算法，为用户提供了极大的方便。另外，该工具箱中还包含了大量的演示实例，有助于初学者加深理解。

1982 年，Hopfield 用能量函数的思想形成了一种新的计算方法，该计算方法由含有对称突触连接的反馈网络执行。而且他还将该反馈网络同用于统计物理的 Ising 模型相类推，这种类推为大量的物理学理论和许多的物理学家进入神经网络领域铺平了道路。Hopfield 阐明了 NN 与动力学的关系，并用非线性动力学的方法来研究这种 NN 的特性，建立了 NN 稳定性判据，并指出信息存储在网络中 NN 之间的连接上，形成了 Hopfield 网络。这是 NN 研究的突破性进展。

此后，神经网络领域的研究有了新发展，在 20 世纪的最后 10 年，产生了大量关于神经网络的论文，并在许多领域应用了神经网络技术，新的理论和实践工作层出不穷，20 世纪 90 年代初始，Vapnik 和合作者发明了一类计算功能强大的导师学习网络，用于解决模式识别、回归及密度估测问题，这种网络被称为支持向量机（Support Vector Machines, SVM），以有限样本学习理论的结论为基础。支持向量机的新特征在于 Vapnik-Chervonenkis（VC）维特征蕴含在向量机的设计中，VC 维数为衡量神经网络样本学习能力提供了一种有效的量度。总之，以 Hopfield 教授 1982 年发表的论文为标志，掀起了神经网络的研究热潮。

本书共分 10 章。其中第 1 章是神经网络的绪论，介绍了神经网络的发展历程、神经网络模型、人工神经网络的分类等内容；第 2 章是 MATLAB 语言及神经元，介绍了 MATLAB 的语言特点、MATLAB 神经网络工具箱、MATLAB 神经网络工具箱的对象与属性等内容；第 3 章是神经网络工具箱函数的分析及实例，介绍了神经网络的应用函数、神经网络的输入函数及其导函数等内容；第 4 章是前向型神经网络及 MATLAB 应用举例，介绍了感知器、线性神经网

络、BP 网络、径向基网络、GMDH 网络等内容；第 5 章是自组织神经网络及 MATLAB 程序，介绍了自组织竞争型神经网络、自组织特征映射神经网络、自适应共振理论模型、对向传播网络等内容；第 6 章是反馈神经网络及其应用，介绍了 Elman 神经网络及 MATLAB 程序、Hopfield 网络及 MATLAB 程序等内容；第 7 章是图形用户接口，介绍了图形用户界面、图形用户应用、数据的存储和读取等内容；第 8 章是 Simulink，介绍了 Simulink 交互式仿真集成环境、Simulink 的基本模块、Simulink 的命令行仿真技术、Simulink 应用实例等内容；第 9 章是自定义神经网络，介绍了自定义神经网络、自定义神经网络的初始化、训练与仿真等内容；第 10 章是神经网络的应用，介绍了线性神经网络在线性预测中的应用、神经模糊控制在洗衣机中的应用、模糊神经网络在配送中心选址中的应用等内容。

本书可作为高等院校计算机、电子工程、控制工程、信息与通信科学、数学、机械工程和生物医学工程等专业学生的参考教材，对从事上述领域工作的广大科技人员具有重要的参考价值，对学习神经网络及其仿真技术的读者来说，也是一本极为有用的入门指导书。

由于时间仓促、作者水平和经验有限，书中错漏之处在所难免，敬请读者指正。

编 著 者

2009 年 3 月

目　　录

第 1 章　神经网络绪论

1.1　人工神经网络的介绍

在你阅读本书时，你正在用到一个复杂的生物神经网络，大约有10^{11}个相互连接的神经元帮助你进行阅读、呼吸、思考，完成各种动作，部分神经网络的结构和功能是与生具有的，比如支配呼吸、哭、吮吸等本能动作的功能；而大多数的功能则需要通过后天的学习才能获得。

虽然人们还并不完全清楚生物神经网络是如何进行工作的，但幻想构造一些“人工神经元”，进而将它们以某种方式连接起来，可以模拟“人脑”的某些功能。

1.2　神经网络的发展历程

人工神经网络这项研究始于 20 世纪 40 年代，它的发展经历了起始、萧条和兴盛三个阶段。

1.2.1　神经网络的起始

1943 年，精神病学家和神经解剖学家 McCulloch 与数学家 Pitts 在数学生物物理学会期刊《Bulletin of Mathematical Biophysics》上发表文章，总结了生物神经元的一些基本生理特征，提出了形式神经元的数学描述与结构，即 MP 模型。McCulloch 和 Pitts 描述了一个逻辑微积分的神经网络，这个网络由神经生理学和数学逻辑学组成，他们定义的神经元的正规模型被认为遵循着“全或无”法则。McCulloch 和 Pitts 证明：原则上，在拥有了数量众多的简单单元和适当的神经元连接且运行同步的情况下，所构建的网络能计算任何可计算的函数。MP 的提出兴起了 NN 研究，同时产生了人工智能（Artificial Intelligence，AI）这一学科。

1948 年，Wiener 所著的著名的《Cybernetics》一书出版，提出了控制、通信和统计信号处理的重要概念，在学科之间抓住了统计方法的物理意义。

1949 年，生理学家 D.O.Hebb 出版了《The Organization of Behavior》一书。该书第一次鲜明地提出了改变神经元连接强度的 Hebb 规则，特别是 Hebb 提出脑中互连信息随着感官学习任务的不同而不断变化，这种变化产生了神经集合。他认为学习过程是在突触（Synapse）上发生的，突触的连接强度随其前后神经元的活动而变化。根据这一假设提出的学习规则为神经网络的学习算法奠定了基础，使神经网络的研究进入了一个重要的发展阶段。

1952 年，Ashby 所著的《Design for a Brain:The Origins of Adaptive Behavior》一书出版。书中论述了“自适应行为不是天生的，而是学习的结果”这一基本概念，而且通过对动物行为（系统）的学习会有更好的改变，书中还强调了类似机器的生物动态和相关的稳定性概念。

1954 年，Minsky 在 Princeton 大学撰写了一篇题为《Theory of Neural-Analog Reinforcement Systems and Its Application to the Brain-Model Problem》的神经网络博士论文。1961 年，Minsky 写了一篇关于早期人工智能的优秀论文，题为《Steps Toward Artificial Intelligence》，论文的后

半部包含了当今神经网络的大部分内容；1976 年，Minsky 出版了《Computation:Finite and Infinite Machines》一书，该书清晰地扩展了 McCulloch 和 Pitts 1943 年的成果，并将其归入自动机和计算理论中。

1954 年，通信理论的先驱和全息照相术的发明者 Gabor 提出了非线性自适应滤波思想，他希望与合作者一起发明一种机器，通过将一个随机过程的样本输入机器中，连同目标函数，实现机器学习。

Von Neumann 是 20 世纪前半叶科学领域中最伟大的人物之一，数学计算机的基础设计就是以他的名字命名的。早在 1949 年，Von Neumann 在 Illinois 大学四次讲座的第二次讲座中，阐述了 McCulloch、Pitts 证实的神经网络理论特点。1955 年，他应邀去 Yale 大学进行 Silliman 讲座，直至 1956 年（他死于 1957 年）。他未完成的 Silliman 讲座的手稿于 1958 年作为一本书出版，书名为《The Computer and the Brain》，此书由于涉及了 Von Neumann 生前所做的工作和他注意的人脑与计算机的巨大差异，因而备受关注。此外，1956 年 Von Neumann 用约简的思想解决了一个在神经网络中特别令人关注的问题，这就是如何用认为是不可靠的神经元来设计一个可靠的网络的问题。

1957 年，Rosenblatt 提出了感知器（Perceptron）的概念。1958 年 Rosenblatt 基于对感知器的研究，提出了解决模式识别问题的新监督学习方法，并证明了所谓的感知器收敛定理，首次把神经网络的研究付诸工程实践。这是一种学习和自组织的心理学模型，它基本上符合神经生理学的理论，模型的学习环境是有噪声的，网络构造中存在随机连接，这是符合动物学习的自然环境的。这种类型的机器显然有可能应用于模式识别（Pattern Recognition）、联想记忆（Associative Memory）等方面。

1960 年，Widrow 和 Hoff 引入了最小均方差（Least Mean-Square, LMS）算法，并用它系统地阐明了自适应线性元件（Adaptive Linear Element）。感知器和自适应线性元件之间的差异在于训练过程，最早出现的包含多适应元件的可训练分层神经网络是多学习机结构，这种结构由 Widrow 和他的学生在 1962 年提出。

1965 年，Nilsson 所著的《Learning Machines》一书出版，至今这本书仍然是关于超平面中线性模型的佳作。

1967 年，Amari 用推测梯度方式进行自适应模式分类。

在感知器盛行的 20 世纪 60 年代，人们对神经网络的研究过于乐观，认为只要将这种 NN 互连成一个网络，就可以解决人脑思维的模拟问题。因此，当时有上百家实验室纷纷投入这项研究，美国军方也投入了巨额资金，当时 NN 在声呐信号识别等领域的应用取得了一定的成绩。

1969 年，Minsky 和 Papert 所著的《Perceptron》一书出版，该书从数学角度证明了关于单层感知器的计算具有根本的局限性，指出感知器的处理能力有限，甚至连异或这样的问题也不能解决，并在多层感知器的总结一章中，论述了单层感知器的所有局限性在多层感知器中是不可能被全部克服的。当时人工智能的以功能模拟为目标的另一分支出现了转机，产生了以知识信息处理为基础的知识工程（Knowledge Engineering），给人工智能从实验室走向实用带来了希望。同时，随着微电子技术的进步，以及传统的 Von Neumann 型数学计算机的发展，使整个学术界陶醉于数学计算机的成功之中，从而使 NN 的研究进入了萧条时期。

1.2.2 神经网络的萧条

根据 Cowan 1990 年提出的观点，有三个原因导致了神经网络研究的十多年停滞。原因之

一是技术上的——没有个人计算机和工作站进行实验，如 Gabor 发展了他的非线性学习滤波器，却花费了额外的六年时间建造了含有类推装置的滤波器；原因之二一半是心理上的，即 Minsky 和 Papert 对感知器的悲观结论，一半是资金上的，即没有代理商资助；原因之三是神经网络和晶格旋转之间的类推还未成熟，直至 1975 年才由 Sherrington 和 Kirkpartrick 创建出旋转镜片模型。

这些因素导致了 20 世纪 70 年代对神经网络的研究陷入了低谷，此时许多研究人员放弃了除心理学和神经学之外的其他领域的研究。

难能可贵的是，在此期间，仍有不少学者在极端艰难的条件下，保持对神经网络的信奉，致力于这一研究。

1972 年 Teuvo Kohonen 和 James Anderson 各自独立发展了用于记忆的新神经网络。Amari 独立提出了一个神经元的附加模型，并将其应用于研究随机连接类似于神经元元件的动态行为之中。Wilson 和 Cowan 从含有兴奋和抑制神经元空间局部化模型的动态行为中获取非线性微分方程组。

1976 年 Willshaw 和 Malsburg 发表了第一篇受人脑拓扑次序映射启发，构筑自组织映射的论文。

1977 年 Anderson、Silverstein、Ritz 和 Jones 提出了黑箱脑状态（Brain-State-in-a-Box, BSB）模型，其中包括含有非线性动力的一个简单联想网络。

1980 年 Grosserg 在早期对竞争学习研究的基础上，创立了自组织新理论，被称为自适应性谐振理论（Adaptive Resonance Theory，ART），该理论包括自下而上的认知层和自上而下的生成层，若输入模式与学习反馈模式相匹配，则产生称为“自适应性谐振”的动力状态（即神经网络的扩大与延长），其前向/后向投影原理已经被其他学者应用于不同方面，并使其得到了发展。

由于新思想的缺乏和用于实验的高性能计算机的短缺，导致了 20 世纪 60 年代后期神经网络研究的停滞不前，到了 20 世纪 80 年代，上述难关均被攻克，神经网络有了迅猛发展，功能日渐强大的个人计算机和工作站开始被广泛应用，此时也产生了神经网络研究的新观念。

使神经网络重新焕发生机的原因，主要有两个，第一个是利用统计机制解释循环网络的运行过程，这种机制可用于联想记忆，此观念是由物理学家 John Hopfield 在他的核心论文中提出的。

20 世纪 80 年代的第二个核心发展是用于训练多层感知器的反向传播算法，此算法同时被几个学者所发现，其中影响最大并广泛传播的是由 David Rumelhart 和 James McClelland 提出的算法，该算法是对 20 世纪 60 年代 Minsky 和 Papert 的批判的一个有力回答。

另外，20 世纪 70 年代后期出现以下情况：

（1）研究视、听觉的人工智能专家认为计算机一般不能从现实世界的实例与现象中获取并总结出知识，也就是说计算机不具备学习能力。于是人们开始意识到了 Von Neumann 体系结构的局限性，转而研究数据流机和并行计算机体系结构。

（2）日本的第五代计算机计划远未达到预想水平，使人们觉得有必要弄清楚人们习以为常的认知功能是如何进行的，这些认知功能包括视听觉感知、学习记忆、运行控制等，从而使人们认识到必须开拓新的思路，探索新的实现途径——与人脑的生理组织更为接近的 NN 模型自然成为理想的候选模型。

（3）在人类智能行为研究方面，神经生理学家、心理学家和计算机科学家相互结合，认为人脑是一个功能十分强大、结构异常复杂的信息系统，但其基本仍是神经元及其之间的连接。

（4）NN 的研究依靠众多学科的共同发展，是多学科的综合产物。而当时许多学科都有了相应的发展，如普里高津（Prigogine）提出了非平稳系统的自组织理论，获诺贝尔奖；哈肯（Haken）研究了大量单元集团运动而产生的宏观效果；非线性系统“混沌”态的提出及其研究，等等。这些都是研究如何通过单元之间的相互连接作用建立复杂系统，类似于生物系统的自组织行为。

（5）脑科学与神经科学的研究成果，迅速反映到神经网络的改进上，例如视觉研究中发现的侧抑制原理、感受野的概念，听觉通道上神经元的自组织排列等。生物的 NN 研究成果对 ANN 的研究起了重要的推动作用。

所有这些重新引起了人们对 ANN 的研究兴趣。

1.2.3 神经网络兴盛

学术界公认，标志 NN 研究高潮的又一次到来是美国加州理工学院生物物理学家 J.Hopfield 教授于 1982 年和 1984 年发表在美国科学院院刊上的两篇文章以及 1986 年 Rumelhart 与 McClelland 的著作。

1982 年，Hopfield 用能量函数的思想形成了一种新的计算方法，该计算方法由含有对称突触连接的反馈网络执行，而且他还将该反馈网络同用于统计物理的 Ising 模型相类推，这种类推为大量的物理学理论和许多的物理学家进入神经网络领域铺平了道路。Hopfield 阐明了 NN 与动力学的关系，并用非线性动力学的方法来研究这种 NN 的特性，建立了 NN 稳定性判据，并指出信息存储在网络中 NN 之间的连接上，形成了 Hopfield 网络。这是 NN 研究的突破性进展。

1982 年，另一个重要的发展是 Kohonen 的关于自组织图的论文发表，其中用到了一个一维或二维的晶体结构，这种模型已成为衡量在此领域中有价值创新的基准。

1983 年，Kirkpatrick、Gelatt 和 Vecchi 提出了模拟退火（Simulated Annealing, SA）的新方法，该方法以统计理论为基础，用于解决组合最优问题。

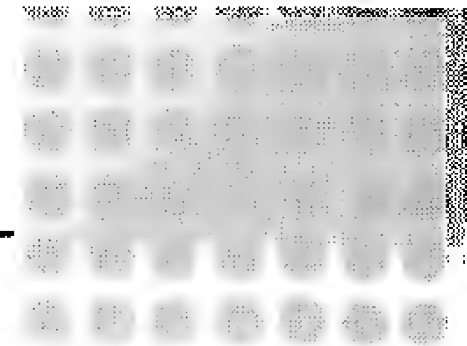
1983 年，Barto、Sutton 和 Anderson 发表了关于强化学习（Reinforcement Learning, RL）的论文，虽然他们不是第一个提出强化学习思想的人（如 Minsky 在 1954 年的博士论文中就有提及），但这篇论文将强化学习应用于实际，验证了其应用有可行性，对强化学习的发展及应用产生了重要的影响。

1984 年，Hopfield 设计与研制了他所提出的神经网络模型的电路，指出神经元可以用运算放大器来实现，所有神经元的连接可用电子线路来模拟。同时，他们进行了神经网络应用研究，成功地解决了旅行商（TSP）计算难题（优化问题），使神经网络的研究步入了兴盛时期。

1984 年，Braitenberg 出版了《Vehicles:Experiments in Synthetic Psychology》一书，此书中引用面向对象技术和自组织原理，通过对假定的基本机制的合成，而不是自顶向下的分析，使得对复杂过程的理解达到最好效果。Braitenberg 通过对各种具有简单内部结构的机器的描述证明了这一条重要法则。Braiterberg 直接和间接地研究动物大脑超过 20 年，在这方面的研究启发其发现机器特性及其行为。

1985 年，Ackley、Hinton 和 Sejnowski 以模拟退火思想为基础，对 Hopfield 模型引入了随机机制，提出了 Boltzmann 机，第一次成功实现了多层神经网络的功能，打破了人们心理上的局限，证明了 Minsky 和 Papert 1969 年的推测根据是不正确的。

1986 年，Rumelhart、Hinton 和 Willianms 发展了反向传播算法（Back-Propagation algorithm,



BP）。同年，Rumelhert 和 McClelland 编写的名为《Parallel Distributed Processing：Explorations in the Mirostructures of Cognition》的书出版，此书的出版对反向传播算法的应用产生了重要影响。反向传播算法已成为大多数多层感知器训练所采用的流行学习算法。该算法解决了多层 NN 的学习问题，证明了多层神经网络的计算能力并不像 Minsky 等人所预料的那样弱，相反它可以完成许多学习任务，解决许多实际问题。

1988 年，Linsker 在感知器网络上提出了一种新的自组织理论，该理论用于保持输入行为模式的最大信息，并受突触连接和活动范围的限制，在 Shanno 信息论的基础上，形成了最大互信息理论。Linsker 的论文重新点燃了基于神经网络的信息应用理论的光芒，尤其对 Bell 和 Sejnowski 提出的盲源分离问题（Blind Source Separation Problem）的信息应用理论产生了影响，同时激起许多学者去探索用其他信息理论模型去解决各自广泛领域中的问题，这种方法称为盲反卷积（Blind Deconvolution）。

1988 年，Broomhead 和 Lowe 用径向基函数（Radial Basis Functions, RBF）提出了分层反馈网络设计的方法。Broomhead 和 Lowe 的论文将神经网络的设计与数值分析的重要领域和线性适应滤波挂钩，产生了大量的研究成果，为多层感知器的研究提供了又一方法和途径。1990 年，Poggio 和 Girosi 应用 Tikhonov 规则理论进一步丰富了径向基理论。

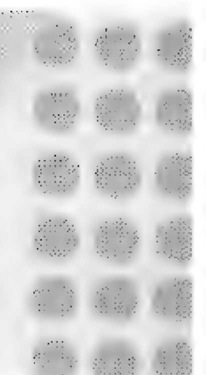
1989 年，Mead 的《Analog VLSI and Neural System》一书出版，这本书将生物神经和集成电路结合在一起，给出了硅视网膜和硅耳蜗。

此后，神经网络领域的研究有了新发展，在 20 世纪的最后 10 年，产生了大量关于神经网络的论文，并在许多领域应用了神经网络技术，新的理论和实践工作层出不穷，20 世纪 90 年代初，Vapnik 和合作者发明了一类计算功能强大的导师学习网络，用于解决模式识别、回归及密度估测问题，这种网络被称为支持向量机（Support Vector Machines, SVM），以有限样本学习理论的结论为基础。支持向量机的新特征在于 Vapnik-Chervonenkis（VC）维特征蕴含在向量机的设计中，VC 维数为衡量神经网络样本的学习能力提供了一种有效的量度。

总之，以 Hopfield 教授 1982 年发表的论文为标志，掀起了神经网络的研究热潮。1987 年 6 月，在美国加州举行了第一届 NN 国际会议，有 1000 多名学者参加，并成立了国际 NN 学会，以后每年召开两次国际联合 NN 大会（IJCNN），其他国际学术会议也都列有 NN 主题。较有影响的国际学术刊物有：《IEEE Transaction on Neural NetWork》和《Neural Network》。美国 IBM、AT&T、贝尔实验室、神经计算机公司、各高校、美国政府制定了“神经、信息、行为科学（NIBS）”计划，投资 5.5 亿美元作为第六代计算机的基础研究；美国科学基金会（NSF）、海军研究局（ONR）和空军科学研究部（AFOSR）三家投资 1000 万美元；美国国防部 DARRA 认为“NN 是解决机器智能的唯一希望”，“这是一项比原子弹工程更重要的技术”，并投资 4 亿美元。主要研究目标为：目标识别与跟踪、连续语音识别、声呐信号辨识。

日本的富士通、日本电气、日立、三菱、东芝等公司也急起直追。1988 年日本提出了所谓的人类尖端科学计划（Human Frontier Science Program），即第六代计算机研究计划。法国提出了“尤里卡”计划，还有德国的“欧洲防御”和前苏联的“高技术发展”等。

我国于 1989 年在北京召开了一个非正式的 NN 会议；1990 年 12 月在北京召开了中国 NN 大会；1991 年在南京成立了中国 NN 学会，由国内 15 个一级学会共同发起“携手探智能，联盟攻大关”的“863”高技术研究计划；自然科学基金、国防科技预研基金也都列入了 NN 研究内容。



1.3 神经网络模型

1.3.1 生物神经元模块

在人类大脑皮层中大约有 100 亿个神经元，60 万亿个神经突触以及它们的连接体。单个神经元处理一个事件需要10^{-3}s，而在硅芯中处理一事件只需要10^{-9}s。但人脑是一个非常高效的结构，大脑中每秒每个动作的能量约为10^{-16}J，而当今性能最好的计算机进行相应的操作需要10^{-6}J。

神经元是基本的信息处理单元。生物神经元主要由树突、轴突和突触组成。其结构示意图如图 1-1 所示。

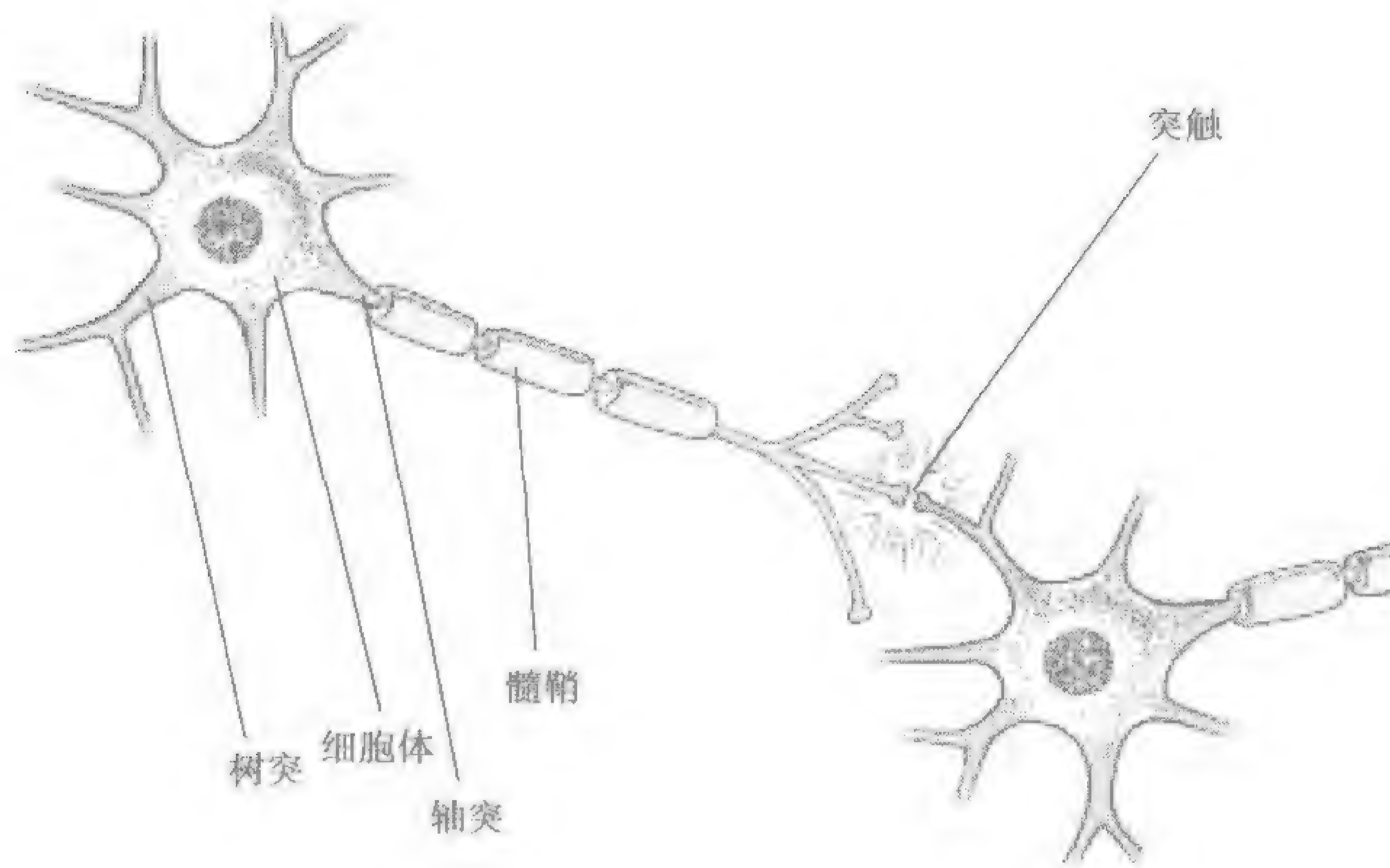


图 1-1　神经元细胞

其中树突是由细胞体向外伸出的，有不规则的表面和许多较短的分支。树突相当于信号的输入端，用于接受神经冲动。轴突是由细胞体向外伸出的最长的一条分支，即神经纤维，相当于信号的输出电缆，其端部的许多神经末稍为信号输出端子，用于传出神经冲动。神经元之间通过轴突（输出）和树突（输入）相互联结，其接口称为突触。每个细胞约有 103 或 104 个突触。神经突触是调整神经元之间相互作用的基本结构和功能单元，最通常的一种神经突触是化学神经突触，它将得到的电信号转化成化学信号，再将化学信号转化成电信号输出。这相当于双接口设备。它能加强兴奋或抑制作用，但两者不能同时发生。细胞膜内外有电位差，约 20～100mV，称为膜电位。膜外为正，膜内为负。

神经元作为信息处理的基本单元，具有如下重要的功能：

（1）可塑性：可塑性反映在新突触的产生和现有神经突触的调整上，可塑性使神经网络能够适应周围的环境。

（2）时空整合功能：时间整合功能表现在不同时间、同一突触上；空间整合功能表现在同一时间、不同突触上。

（3）兴奋与抑制状态：当传入冲动的时空整合结果使细胞膜电位升高，超过被称为动作电位的阈值（约 40mV）时，细胞进入兴奋状态，产生神经冲动，由轴突输出；同样，当膜电位低于阈值时，无神经冲动输出，细胞进入抑制状态。

（4）脉冲与电位转换：沿神经纤维传递的电脉冲为等幅、恒宽、编码（60～100mV）的离散脉冲信号，而细胞电位的变化为连续信号。在突触接口处进行“数/模”转换。神经元中的轴

突非常长和窄，具有电阻高、电压大的特性，因此轴突可以建模成阻容传播电路。

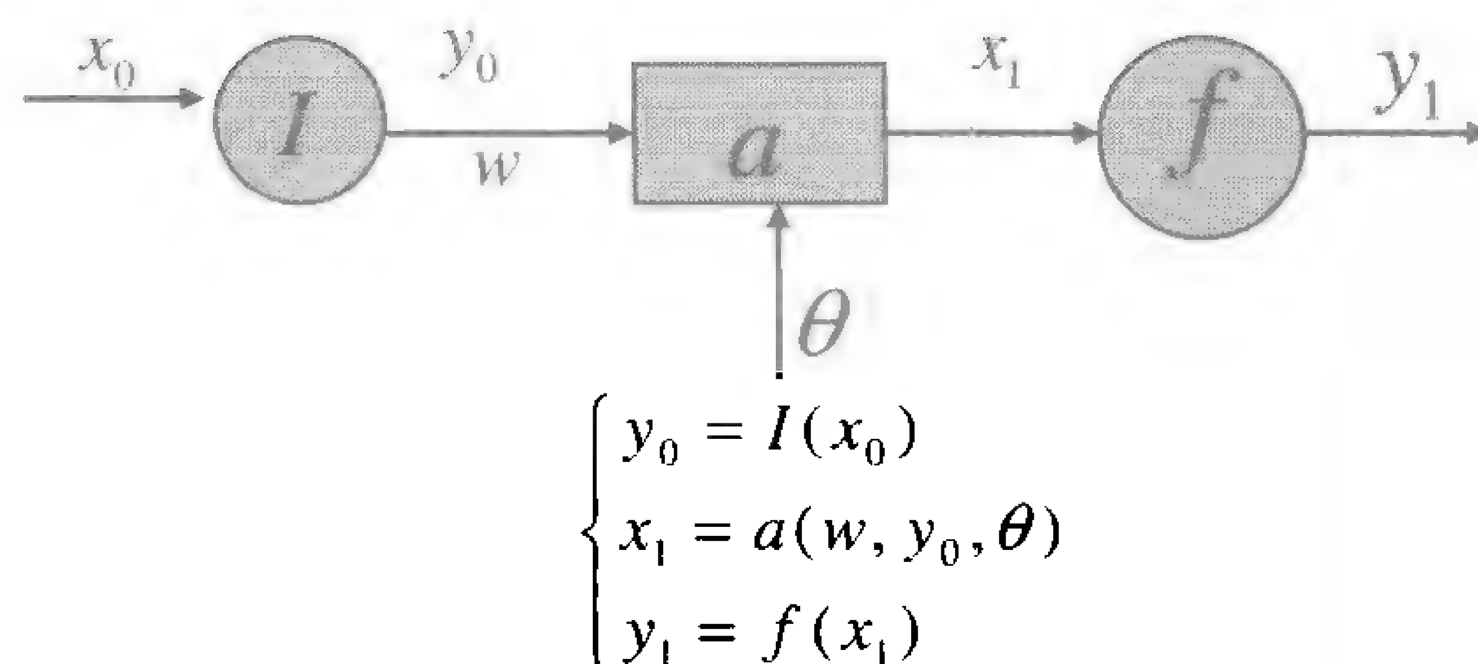
（5）突触的延时和不应期：突触对神经冲动的传递具有延时和不应期，在相邻的二次冲动之间需要一个时间间隔。在此期间对激励不响应，不能传递神经冲动。

（6）学习、遗忘和疲劳：突触的传递作用有学习、遗忘和疲劳过程。

1.3.2　人工神经元模型

1．单输入单输出人工神经元

一个单输入单输出的人工神经元如图 1-2 所示。



$$\begin{cases} y_0 = I(x_0) \\ x_1 = a(w, y_0, \theta) \\ y_1 = f(x_1) \end{cases}$$

图 1-2　单输入单输出的人工神经元

从图 1-2 中可以看出，标量输入 x_0 经过预处理函数单元 I ，预处理单元 I 的输出为

$$y_0 = I(x_0) \tag{1-1}$$

标有 a 的方框称为输入函数或激活函数，其输入/输出关系为

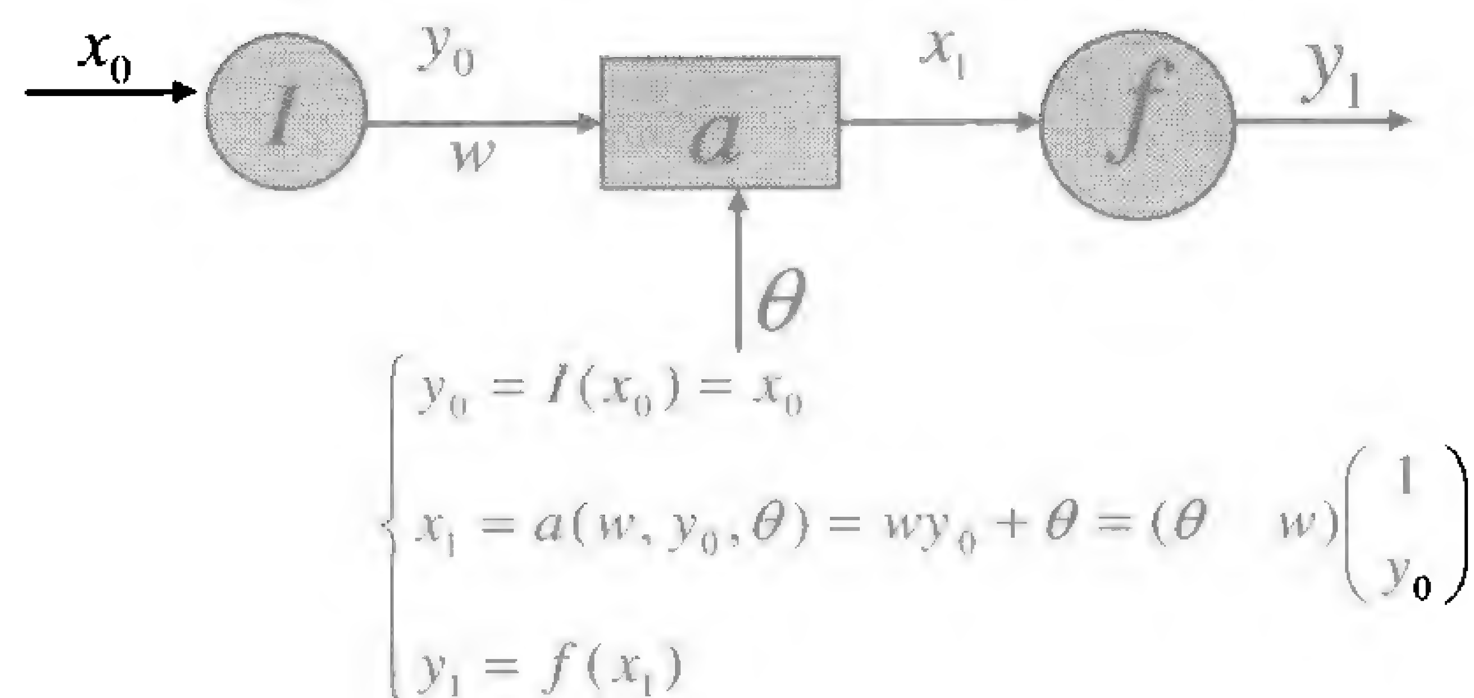
$$x_1 = a(w, y_0, \theta) \tag{1-2}$$

其中输入函数（激活函数）a 可以是任意的函数，w 称为权（或权值，在这里是标量），θ 称为阈值。权值和阈值都是可以调整的参数。如果不想在神经元中使用阈值，可以忽略它。在后面的章节中将会出现这样的情况。输入函数的输出 x_1 通常被称为净输入，它被送入一个变换函数（或传输函数）f ，在 f 中产生神经元的标量输出 y_1 。

标有 f 的圆圈称为人工神经元的变换函数，其输入/输出关系为

$$y_1 = f(x_1) \tag{1-3}$$

在具体应用时，人工神经元的预处理函数、输入函数和变换函数都有具体的形式。这里假定其预处理函数是恒等变换函数，输入函数是内积函数，如图 1-3 所示。



$$\begin{cases} y_0 = I(x_0) = x_0 \\ x_1 = a(w, y_0, \theta) = wy_0 + \theta = (\theta \quad w)\begin{pmatrix} 1 \\ y_0 \end{pmatrix} \\ y_1 = f(x_1) \end{cases}$$

图 1-3　输入函数为内积函数时的单输入单输出人工神经元

根据图 1-3，预处理单元 I 的输入/输出关系为

$$y_0 = I(x_0) = x_0 \tag{1-4}$$

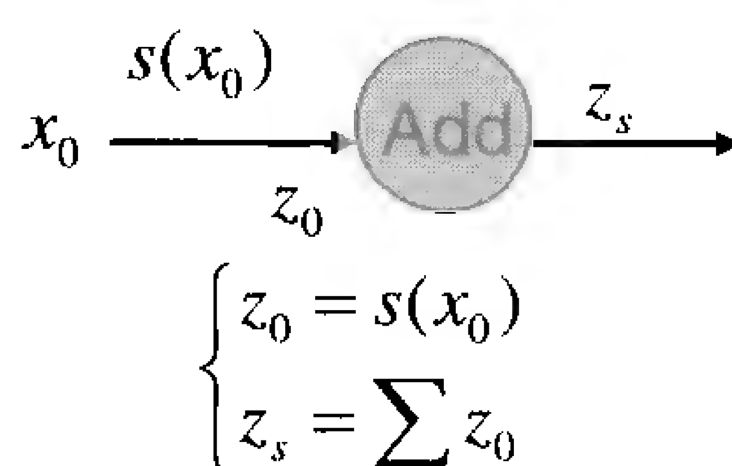
输入函数（或激活函数）的输入/输出关系为

$$x_1 = a(w, y_0, \theta) = wy_0 + \theta = (\theta \quad w)\begin{pmatrix} 1 \\ y_0 \end{pmatrix} \tag{1-5}$$

在本书中，变换函数 $y_1 = f(x_1)$ 的具体形式是类支集函数。

若将这个简单的模型和前面所讨论的生物神经元相对照，则权值 w 对应于突触的连接强度，细胞体对应于输入函数（激活函数）和变换函数，神经元输出 y_1 代表轴突信号。

在此使用样条函数神经元，引入了两类样条函数神经元。样条函数神经元结构简单，第一类样条函数神经元结构图如图 1-4 所示。



$$\begin{cases} z_0 = s(x_0) \\ z_s = \sum z_0 \end{cases}$$

图 1-4　第一类样条函数神经元结构图

在图 1-4 所示的样条函数神经元结构图中，净输入

$$z_0 = s(x_0) \tag{1-6}$$

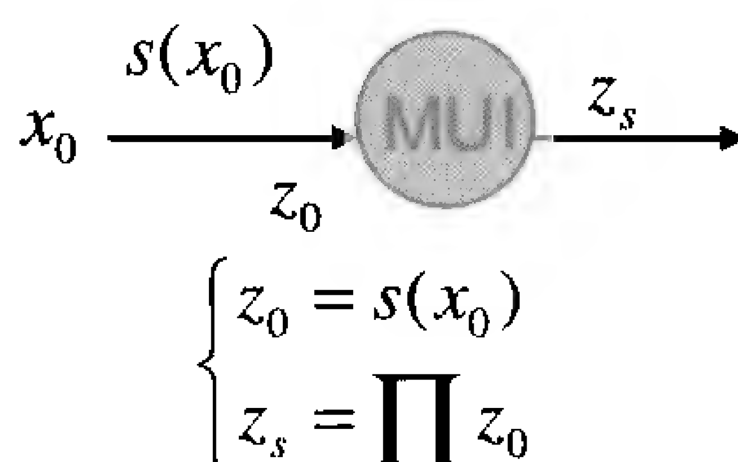
式中（1-6）中 $s(x_0)$ 是自变量为 x_0 的一元样条函数，称为样条权函数。样条权函数的一个重要的特点就是它是输入自变量的函数，而不是传统方法中的常数。图 1-4 中标有 Add 的圆圈表示加法器，它将净输入相加求和，由于图 1-4 的净输入只有一个变量，有

$$z_s = \sum z_0 = z_0 \tag{1-7}$$

在此还引入另外一类样条函数神经元（第二类样条函数神经元），其结构如图 1-5 所示。

与图 1-4 不同，图 1-5 中标有 MUI 的圆圈表示乘法器，它将净输入相乘，由于图 1-5 的净输入只有一个变量，所以有

$$z_s = \prod z_0 = z_0 \tag{1-8}$$



$$\begin{cases} z_0 = s(x_0) \\ z_s = \prod z_0 \end{cases}$$

图 1-5　第二类样条函数神经结构图

2．多输入单输出人工神经元

通常，一个神经元有不只一个输入，具有 m 个输入的一般人工神经元如图 1-6 所示。

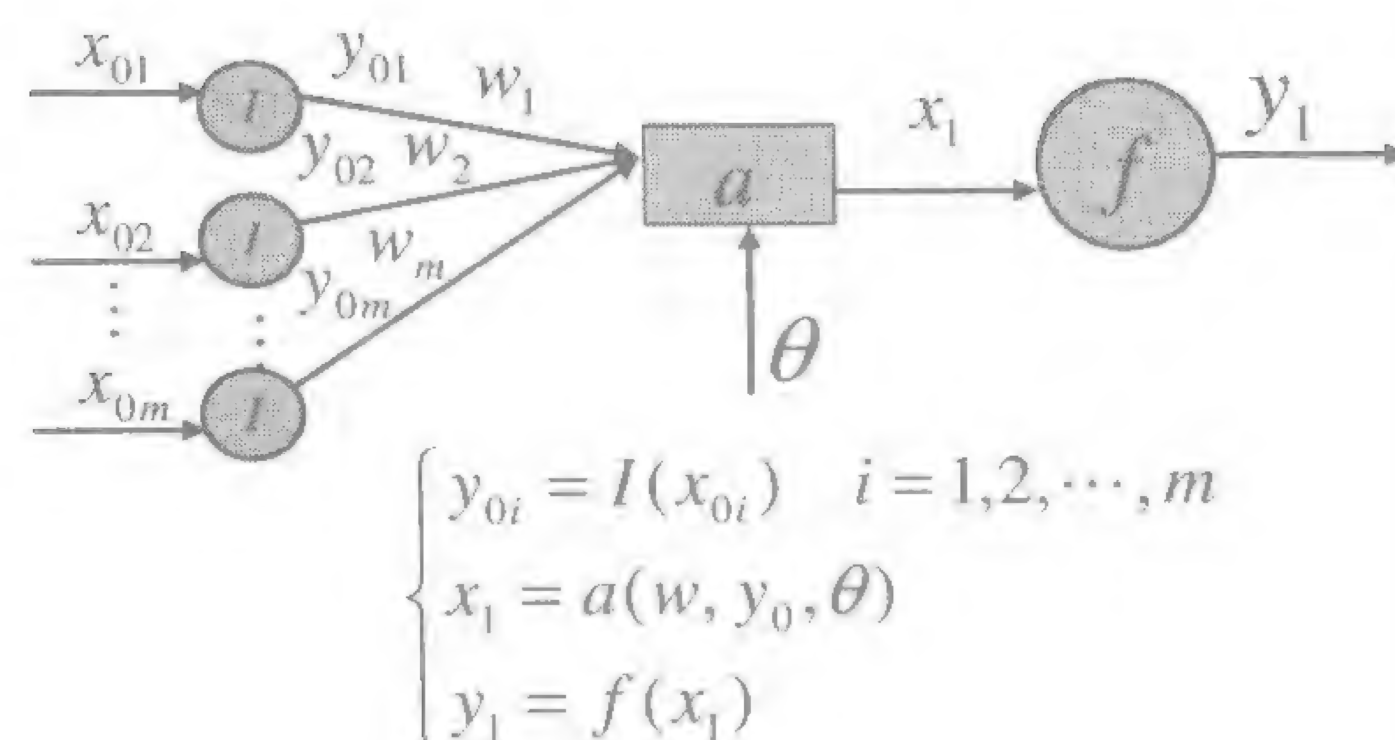


图 1-6　具有 m 个输入，一个输出的一般人工神经元

从图 1-6 中可以看出，每一个标量输入 $x_{0i}(i=1,2,\cdots,m)$ 经过预处理函数单元 I 后，得到的输出为

$$y_{0i} = I(x_{0i}) \tag{1-9}$$

标有 a 的方框是输入函数（或激活函数），其输入/输出关系为

$$x_1 = a(w, y_0, \theta) \tag{1-10}$$

其中

$$\boldsymbol{w} = (w_1, w_2, \cdots, w_m)^{\mathrm{T}} \tag{1-11}$$

$$\boldsymbol{y}_0 = (y_{01}, y_{02}, \cdots . y_{0m})^{\mathrm{T}} \tag{1-12}$$

式（1-10）中输入函数（激活函数）a 可以是任意的函数，$\boldsymbol{w}$ 为权向量，θ 为阈值。净输入 x_1 被送入一个变换函数（或传输函数）f，在 f 中产生神经元的标量输出 y_1。除了输入端增加了维数之外，其余和图 1-2 是一样的。

与前面类似，这里假定其预处理函数是恒等变换函数，输入函数是内积函数，如图 1-7 所示。

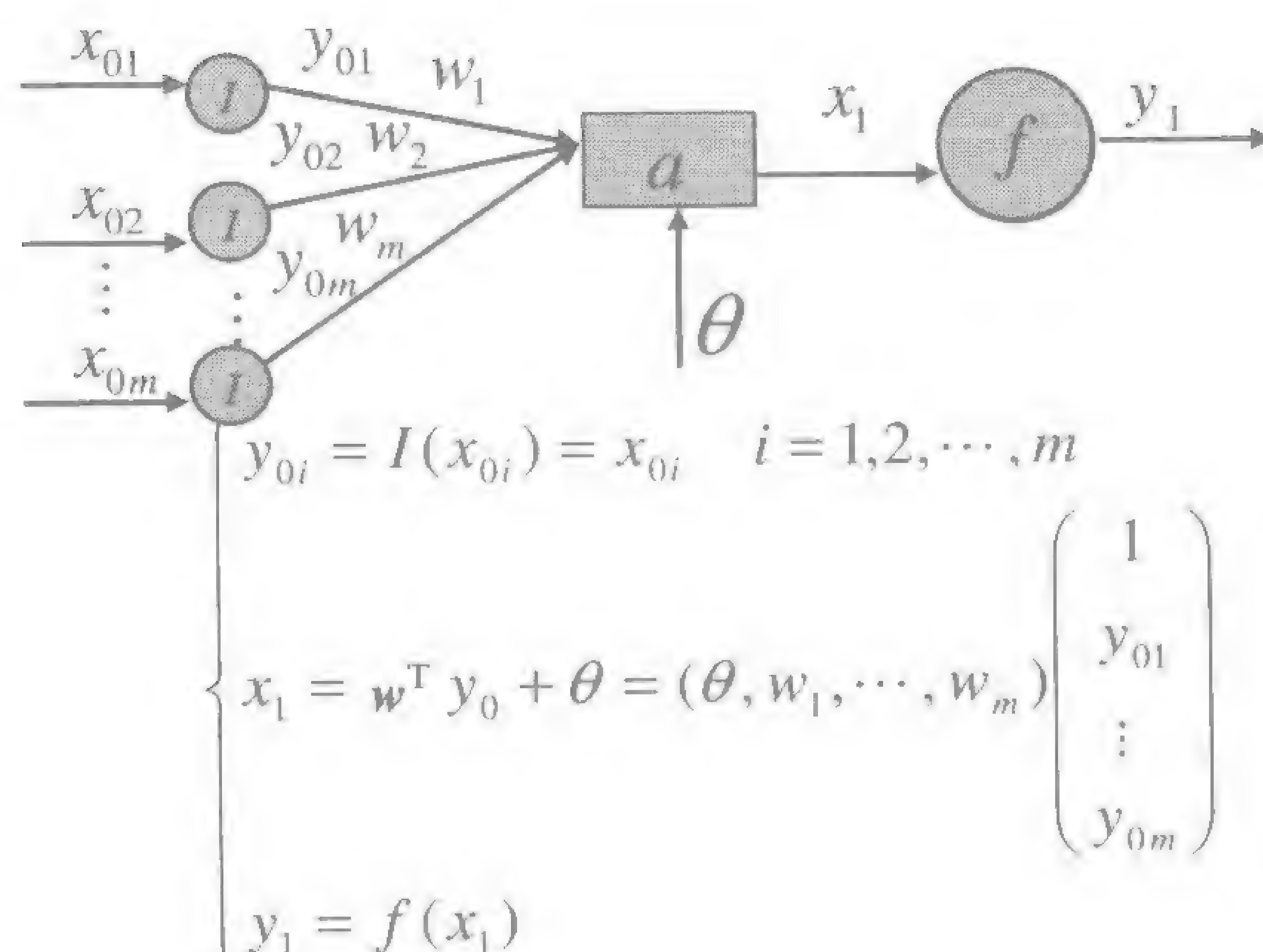


图 1-7　输入函数为内积函数时的多输入单输出人工神经元

根据图 1-7，预处理单元 I 的输入/输出关系为

$$y_{0i}=I(x_{0i})=x_{0i} \qquad i=1,2,\cdots,m \tag{1-13}$$

输入函数（或激活函数）的输入/输出关系为

$$x_1=\boldsymbol{w}^{\mathrm{T}}y_0+\theta=(\theta,w_1,\cdots,w_m)\begin{pmatrix}1\\y_{01}\\\vdots\\y_{0m}\end{pmatrix} \tag{1-14}$$

其中

$$\tilde{\boldsymbol{w}}=(\theta,w_1,\cdots,w_m)^{\mathrm{T}} \tag{1-15}$$

$$\tilde{\boldsymbol{y}}_0=(1,y_{01},\cdots,y_{0m})^{\mathrm{T}} \tag{1-16}$$

$\tilde{\boldsymbol{w}}=(\theta,w_1,\cdots,w_m)^{\mathrm{T}}$ 和 $\tilde{\boldsymbol{y}}_0=(1,y_{01},\cdots,y_{0m})^{\mathrm{T}}$ 分别称为扩充权和扩充输入。

与前面相同，变换函数 $y_1=f(x_1)$ 的具体形式是类支集函数。

第一类多输入单输出样条函数神经元结构如图 1-8 所示。

在图 1-8 所示的第一类多输入单输出样条函数神经元结构中，净输入

$$z_i=s_i(x_i) \qquad i=1,2,\cdots,m \tag{1-17}$$

式（1-17）中 $s_i(x_i)$ 是自变量为 x_i 的一元样条函数，称为样条权函数。由于图 1-8 的净输入有 m 个变量，所以有

$$z_s=\sum_{i=1}^{m}z_i \tag{1-18}$$

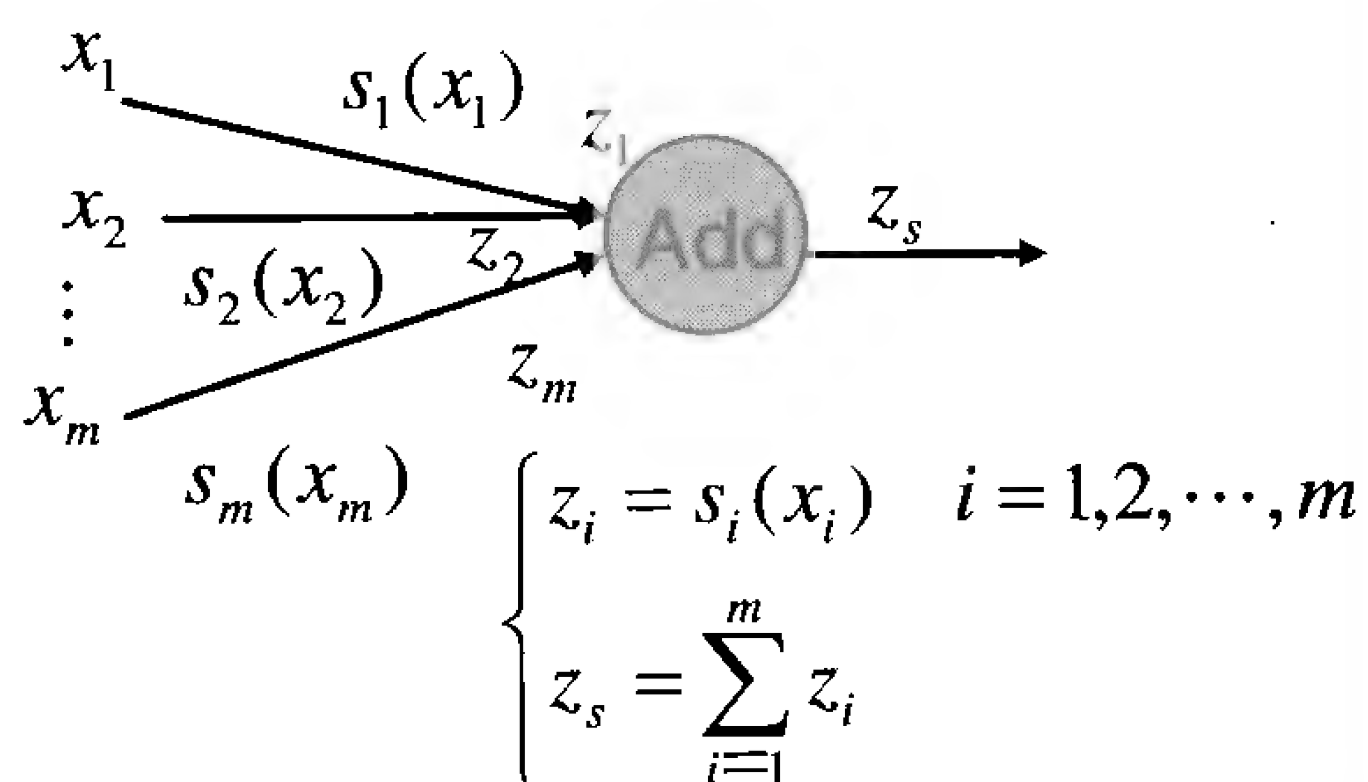


图 1-8　第一类多输入单输出样条函数神经元结构图

对于图 1-9 所示的第二类多输入单输出样条函数神经元结构，净输入仍然由式（1-17）计算，输出为

$$z_s=\prod_{i=1}^{m}z_i \tag{1-19}$$

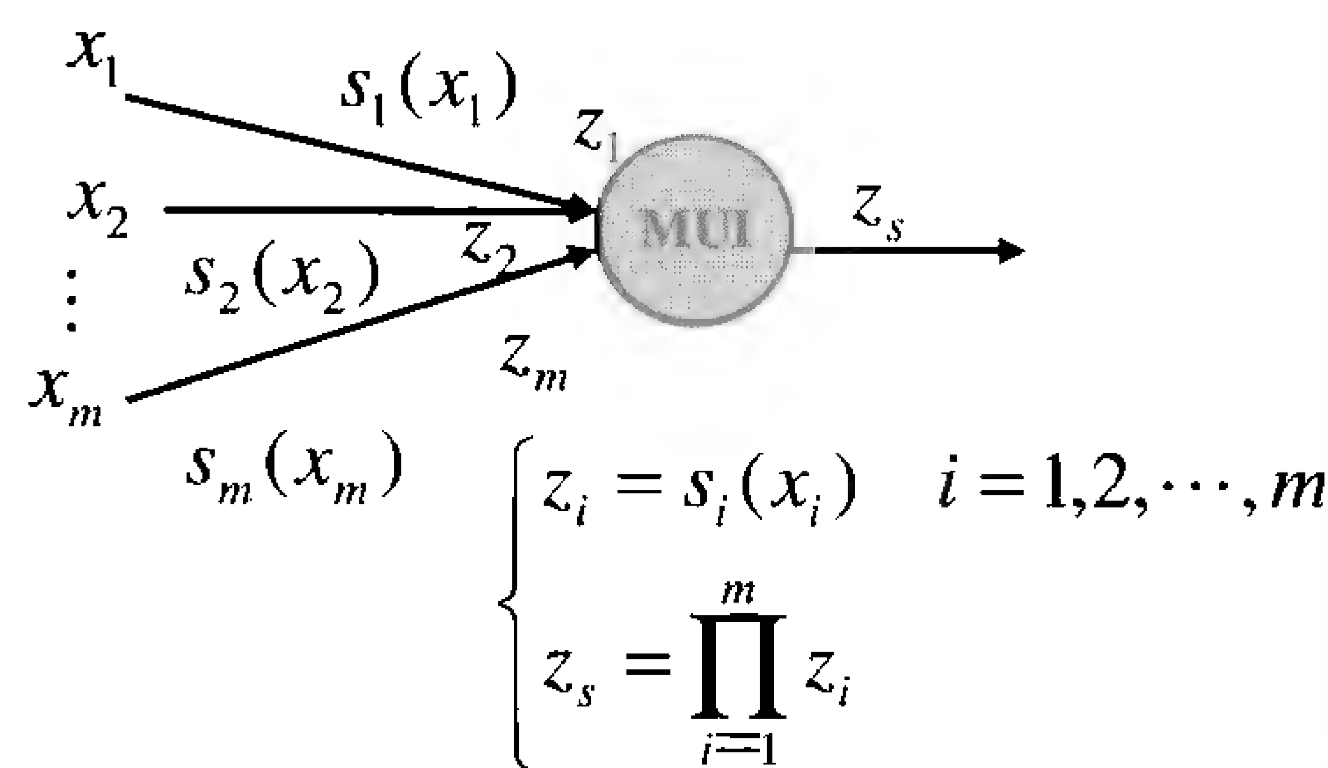


图 1-9　第二类多输入单输出样条函数神经元结构图

1.4　人工神经网络的分类

前面介绍了人工神经元模型，将大量的神经元进行连接可构成人工神经网络。神经网络中神经元的连接方式与用于训练网络的学习算法是紧密结合的，可以认为应用于神经网络设计中的学习算法是被结构化了的。

下面先来介绍人工神经网络的分类。

可以从不同的角度对人工神经网络进行分类，如：

（1）从网络性能角度可分为连续型与离散型网络、确定性与随机性网络。

（2）从网络结构角度可为前向网络与反馈网络。

（3）从学习方式角度可分为有导师学习网络和无导师学习网络。

（4）按连续突触性质可分为一阶线性关联网络和高阶非线性关联网络。

本书将网络结构和学习算法相结合，对人工神经网络进行分类。

1．单层前向网络

所谓单层前向网络是指拥有的计算节点（神经元）是“单层”的，如图 1-10 所示。这里表示源节点个数的“输入层”被看做一层神经元，因为该“输入层”不具有执行计算的功能。

2．多层前向网络

多层前向网络与单层前向网络的区别在于：多层前向网络含有一个或更多的隐含层，其中计算节点被相应地称为隐含神经元或隐含单元，如图 1-11 所示。

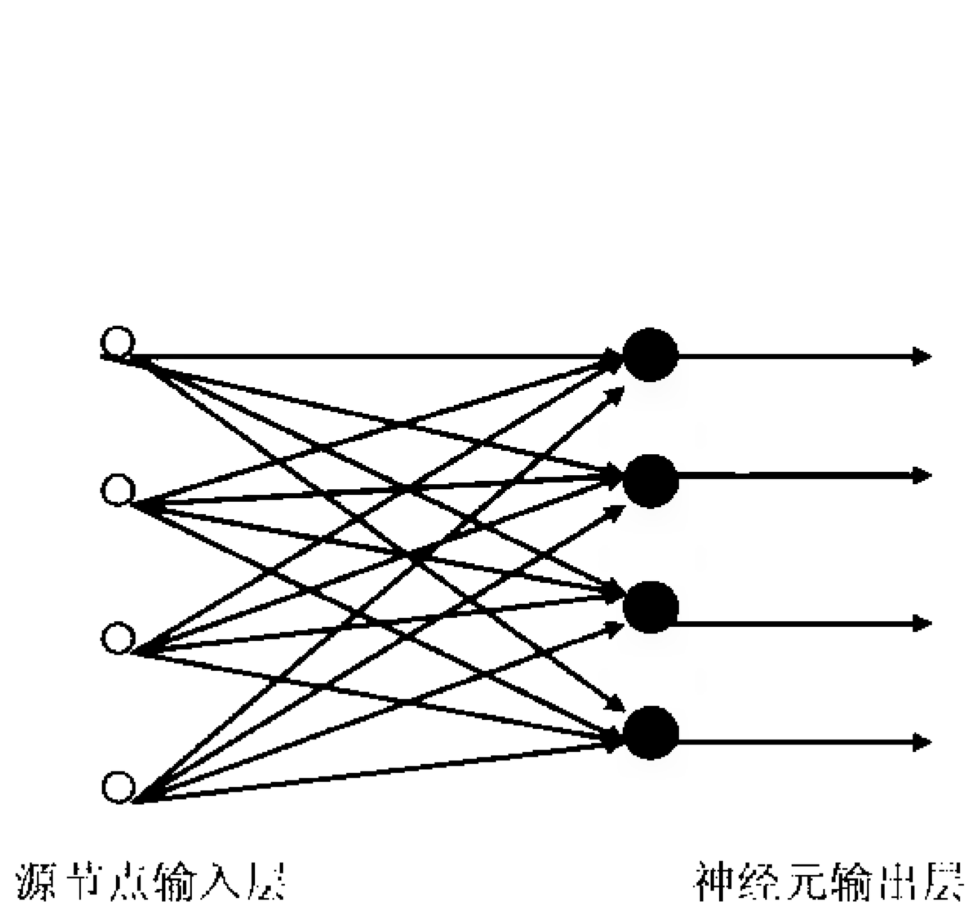


图 1-10　单层前向网络

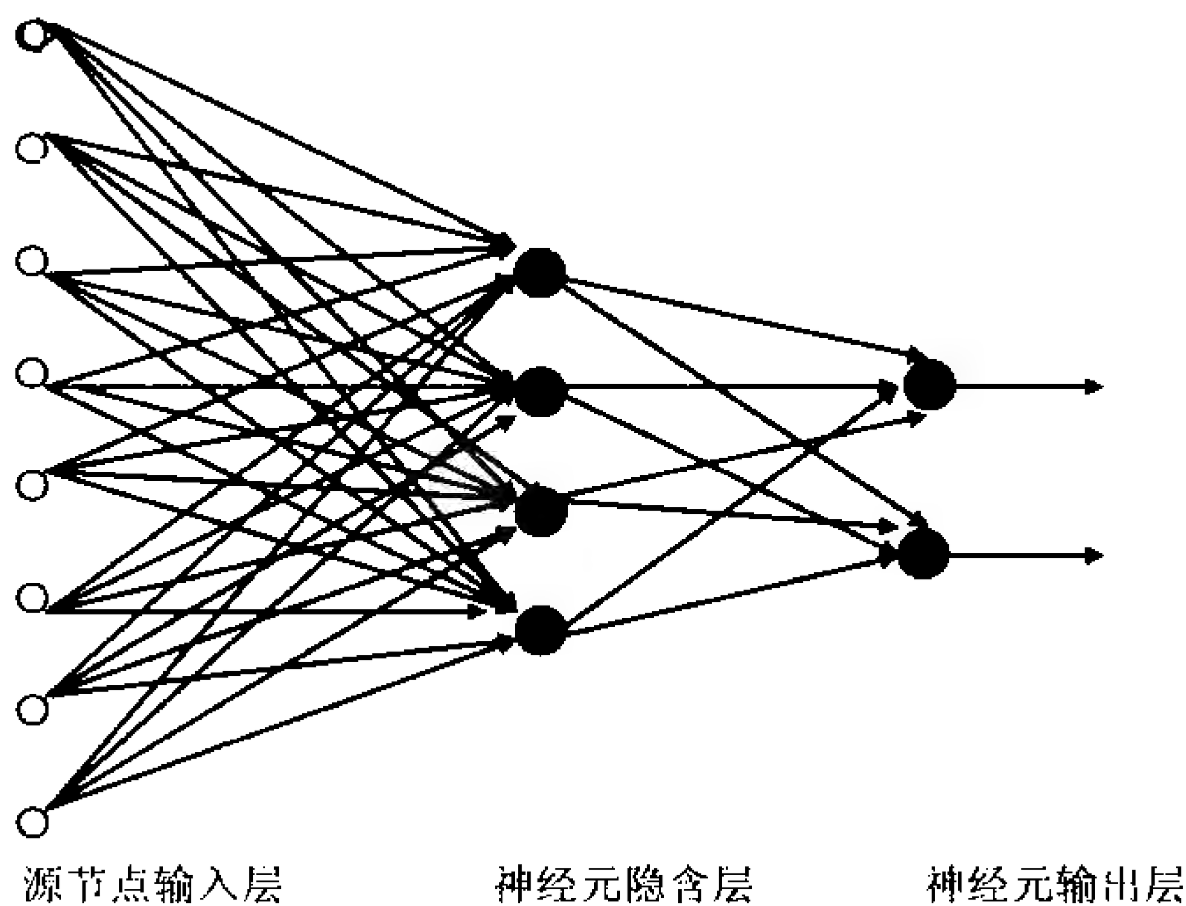


图 1-11　多层前向网络

图 1-11 所示的多层前向网络由含有 8 个神经元的输入层、含有 4 个神经元的隐含层和含有 2 个神经元的输出层所组成。

网络输入层中的每个源节点的激励模式（输入向量）单元组成了应用于第二层（如第一隐含层）中神经元（计算节点）的输入信号，第二层输出信号成为第三层的输入，其余层类似。网络每一层的神经元只含有作为它们输入前一层的输出信号，网络输入层（终止层）神经元的输出信号组成了对网络中输入层（起始层）源节点产生的激励模式的全部响应。即信号从输入层输入，经隐含层传给输出层，由输出层得到输出信号。

通过加入一个或更多的隐含层，使网络能提取出更高序的统计，尤其当输入层规模庞大时，隐含神经元提高高序统计数据的能力便显得格外重要。

3. 反馈网络

所谓反馈网络是指在网络中至少含有一个反馈回路的神经网络。反馈网络可以包含一个单层神经元，其中每个神经元将自身的输出信号反馈给其他所有神经元的输入，如图 1-12 所示，图中所示的网络即为著名的 Hopfield 网络。图 1-13 所示的是另一类型的含有隐含层的反馈网络，图中的反馈连接起始于隐含神经元和输出神经元。图 1-12 和图 1-13 我所示的网络结构中没有自反馈回路。自反馈回路是指一个神经元的输出反馈至其输入，含有自反馈的网络也属于反馈网络。

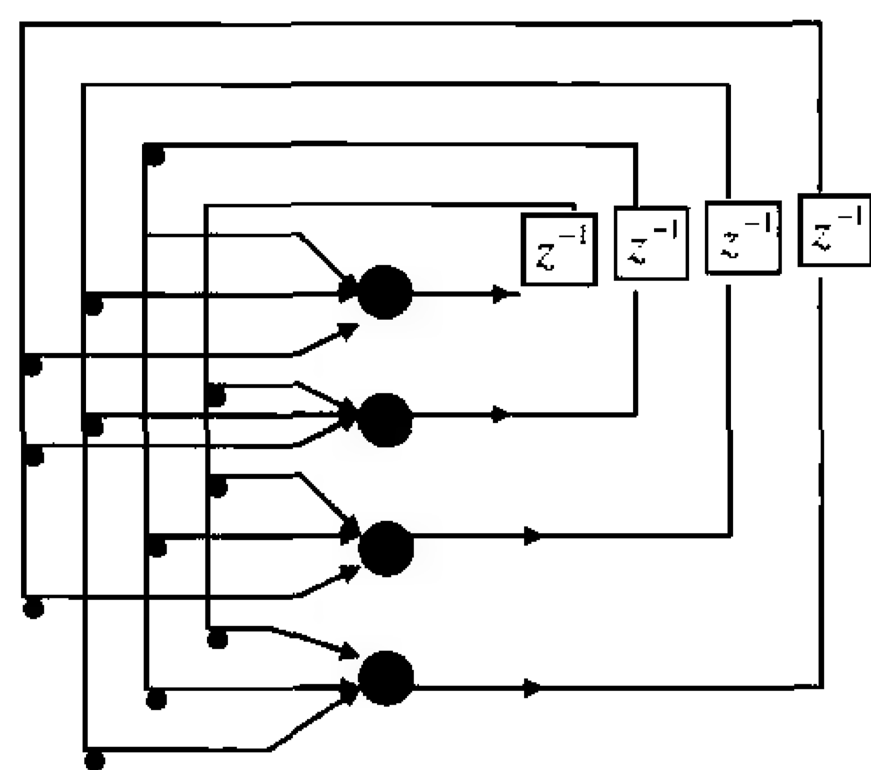


图 1-12　无自反馈和隐含层的反馈网络

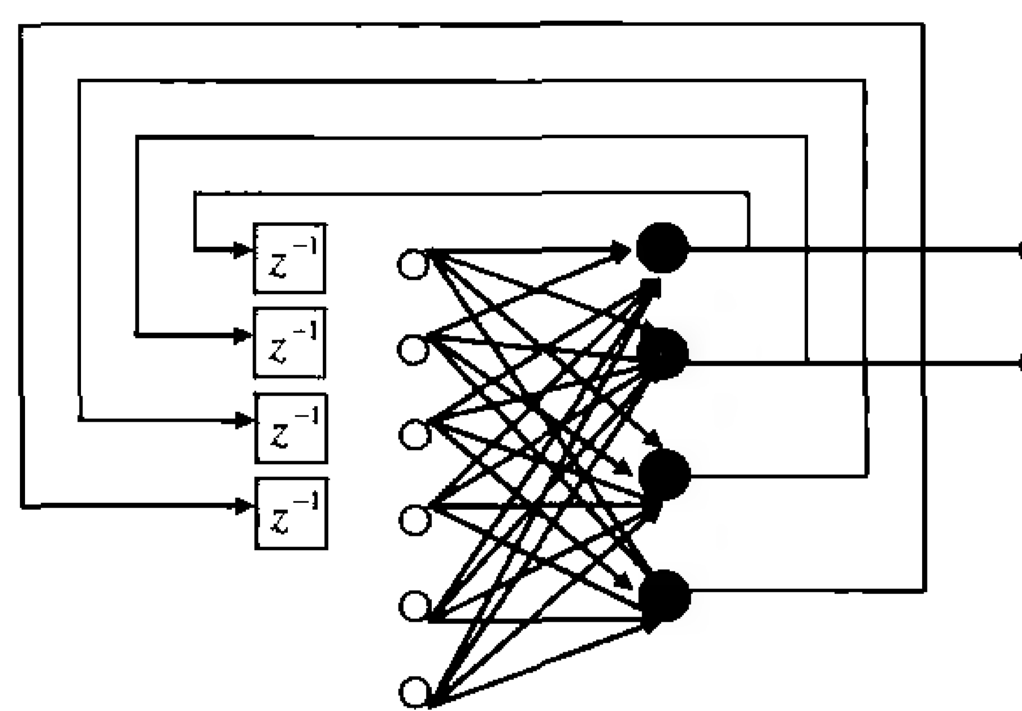


图 1-13　含有隐含层的反馈网络

4. 随机神经网络

随机神经网络是对神经网络引入随机机制，认为神经元是按照概率的原理进行工作的，这就是说，每个神经元的兴奋或抑制具有随机性，其概率取决于神经元的输入。

5. 竞争神经网络

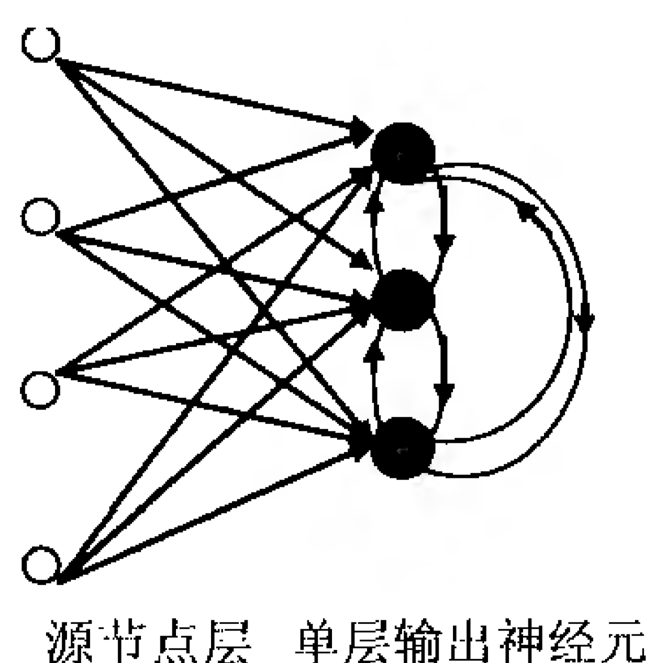


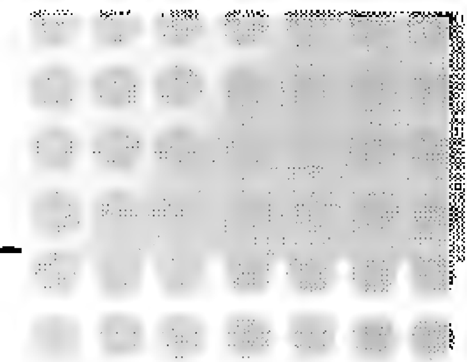
图 1-14　最简单的竞争神经网络——Hamming 网络

竞争神经网络的显著特点是它的输出神经元相互竞争以确定胜者，胜者指出哪一种原形模式最能代表输入模式。

Hamming 网络是一个最简单的竞争神经网络，如图 1-14 所示。神经网络有一个单层的输出神经元，每个输出神经元都与输入节点全相连，输出神经元之间全互连。从源节点到神经元之间是兴奋性连接，输出神经元之间横向侧抑制。

1.5　神经网络的学习方式

神经网络的学习也称为训练，指的是通过神经网络所在环境的刺激作用调整神经网络的自由参数，使神经网络以一种新的方式对外部环境作出反应的一个过程。能够从环境中学习和在学习中提高自身性能是神经网络最有意义的性质。神经网络经过反复学习对其环境更为了解。



学习算法是指针对学习问题的明确规则集合。学习类型是由参数变化产生的形式决定的，不同的学习算法对神经元的突触权值调整的表达式不同。没有一种独特的学习算法用于设计所有的神经网络。选择或设计学习算法时还需要考虑神经网络的结构及神经网络与外界环境相连的形式。

学习方式可分为：有导师学习（Learning With a Teacher）和无导师学习（Learning Without a Teacher）。

（1）有导师学习。

有导师学习又称为监督学习（Supervised Learning），在学习时需要给出导师信号或称为期望输出（响应）。神经网络对外部环境是未知的，但可以将导师看做对外部环境的了解，由输入/输出样本集合来表示。导师信号或期望响应代表了神经网络执行情况的最佳效果，即对于网络输入调整网络参数，使得网络输出逼近导师信号或期望响应。

（2）无导师学习。

无导师学习包括强化学习（Reinforcement Learning）与无监督学习（Unsupervised Learning）或称为自组织学习（Self-Organized Learning）。在强化学习中，对输入/输出映射的学习是通过与外界环境的连续作用最小化性能的标量索引而完成的。在无监督学习或称为自组织学习中没有外部导师或评价来统观学习过程，而是提供一个关于网络学习表示方法质量的测量尺度，根据该尺度将网络的自由参数最优化。一旦网络与输入数据的统计规律性达成一致，就能够形成内部表示方法来为输入特征编码，并由此得出新的类别。

下面介绍 5 个基本的神经网络学习规则：Hebb 学习、基于记忆的学习、纠错学习、竞争学习和随机学习。

1．Hebb 学习

1994 年，Hebb 提出神经网络学习的规则，它是最古老也是最著名的学习规则。为了纪念这位伟大的神经心理学家，将这种学习规则命名为 Hebb 学习规则。

Hebb 学习规则用于调整神经网络的突触权值，可以概括为：

（1）如果一个突触（连续）两边的两个神经元被异步激活，则该突触的能量就被有选择地消弱或消除。

（2）如果一个突触（连续）两边的两个神经元被同时（即同步）激活，则该突触的能量就被选择性地增加。

Hebb 学习规则的数学描述如下：

w_{ij} 表示神经元 x_j 到 x_i 的突触权值，$\overline{x}_j$ 和 $\overline{x}_i$ 分别表示神经元 j 和 i 在一段时间内的平均值，在学习步骤为 n 时对突触权值的调整为

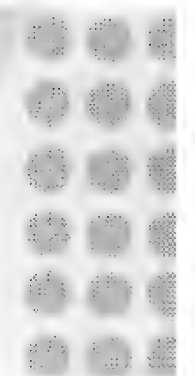
$$\Delta w_{ij}(n)=\eta((x_j(n)-\overline{x}_j)(x_i(n)-\overline{x}_i)) \tag{1-20}$$

式中，η 是正常数，它决定了在学习过程中从一个步骤进行到另一个步骤的学习速率，称其为学习速率。

式（1-20）表示：

① 如果神经元 j 和 i 活动充分时，即同时满足条件 $x_j>\overline{x}_j$ 和 $x_i>\overline{x}_i$ 时，突触权值 w_{ij} 增强。

② 如果神经元 j 活动充分（即 $x_j>\overline{x}_j$ ）而神经元 i 活动不充分（ $x_i<\overline{x}_i$ ），或者神经元 i 活动充分（ $x_i>\overline{x}_i$ ）而神经元 j 活动不充分（ $x_j<\overline{x}_j$ ）时，突触权值 w_{ij} 减小。



2. 基于记忆的学习规则

基本记忆的学习主要用于模式分类，在基于记忆的学习中，过去的学习结果被存储在一个大的存储器中，当输入一个新的测试向量 $\boldsymbol{x}_{\text{test}}$ 时，学习过程就是将 $\boldsymbol{x}_{\text{test}}$ 归到已存储的某个类中。所有基于记忆的学习算法均包括两部分：一是用于定义 $\boldsymbol{x}_{\text{test}}$ 的局部领域的标准；二是用于在 $\boldsymbol{x}_{\text{test}}$ 的局部领域训练样本的学习规则。

一种简单而有效的基于记忆的学习算法就是最近邻规则。设存储器中所记忆的某一类 $\boldsymbol{l}_1$ 含有向量 $\boldsymbol{x}'_N \in \{x_1, x_2, \cdots, x_N\}$，如果式（1-21）成立

$$\min_i d(\boldsymbol{x}_i, \boldsymbol{x}_{\text{test}}) = d(\boldsymbol{x}'_N, \boldsymbol{x}_{\text{test}}) \qquad i = 1, 2, \cdots, N \tag{1-21}$$

则 $\boldsymbol{x}_{\text{test}}$ 属于 $\boldsymbol{l}_1$ 类。其中 $d(\boldsymbol{x}'_N, \boldsymbol{x}_{\text{test}})$ 是向量 $\boldsymbol{x}_i$ 与 $\boldsymbol{x}_{\text{test}}$ 的欧氏距离。

Cover 和 Hart 将最近邻规则作为模式识别的工具加以研究。分析是以以下两个假设作为基础的：

（1）样本 $(\boldsymbol{x}_i, \boldsymbol{d}_i)$ 独立同分布，依照样本 $(\boldsymbol{x}, \boldsymbol{d})$ 的联合概率分布。

（2）样本数量 N 无限大。

在上述假设条件下，由最近邻规则导致的分类错误概率被限制于两倍 Bayes 错误概率之下，也就是所有判定规则中的最小错误概率。

最近邻分类器的变形是 k 阶近邻分类器。其思想为：如果与测试向量 $\boldsymbol{x}_{\text{test}}$ 最近的 k 个向量均是某类别的向量，则 $\boldsymbol{x}_{\text{test}}$ 属于该类别。

3. 纠错学习规则

首先我们考虑一个简单的情况：设某神经网络的输出层中只有一个神经元 i，给该神经网络加上输入，这样就产生了输出 $y_i(n)$，称该输出为实际输出。对于所加上的输入，我们期望该神经网络的输出为 $d(n)$，称为期望输出或目标输出。实际输出与期望输出之间存在着误差，用 $e(n)$ 表示。

$$e(n) = d(n) - y_i(n) \tag{1-22}$$

现在要调整突触权值，使误差信号 $e(n)$ 减小。为此，可设定代价函数或性能指数 $E(n)$

$$E(n) = \frac{1}{2} e^2(n) \tag{1-23}$$

反复调整突触权值使代价函数达到最小或使系统达到一个稳定状态（即突触权值稳定），就完成了学习过程。

该学习过程称为纠错学习，也称为 Delta 规则或者 Widrow-Hoff 规则。

w_{ij} 表示神经元 x_j 到 x_i 的突触权值，在学习步骤为 n 时对突触权值的调整为

$$\Delta w_{ij}(n) = \eta e(n) x_j(n) \tag{1-24}$$

式中，η 为学习速率参数。式（1-24）表明：对神经元突触权值的调整与突触误差信号和输入信号成比例。纠错学习实际上是局部的，Delta 规则所规定的突触权值调整局限于神经元 i 的周围。

得到 $\Delta w_{ij}(n)$ 之后，定义突触权值 w_{ij} 的校正值为

$$w_{ij}(n+1) = w_{ij}(n) + \Delta w_{ij}(n) \tag{1-25}$$

式中，$w_{ij}(n)$ 和 $w_{ij}(n+1)$ 可分别看做是突触权值 w_{ij} 的旧值和新值。

4．竞争学习规则

在竞争学习中，神经网络的输出神经元之间相互竞争，在任一时间只能有一个输出神经元是活动的。而在基于 Hebbian 学习的神经网络中几个输出神经元可能同时是活动的。

竞争学习规则有三项基本内容：

（1）一个神经元集合。除了某些随机分布的突触权值以外，所有的神经元都相同，因此对给定的输入模式集合有不同的响应。

（2）一个机制。允许神经元通过竞争对一个给定的输入子集作出响应。赢得竞争的神经元被称为全胜神经元。

（3）每个神经元的能量都被限制。

在竞争学习的最简单形式中，神经网络有一个单层的输出神经元，每个输出神经元都与输入节点全相连，输出神经元之间全互连，如图 1-14 所示。从源节点到神经元之间是兴奋性连接，输出神经元之间横向侧抑制。

对于一个指定输入模式 x，一个神经元 i 成为获胜神经元，则它的感应局部区域 v_i 大于网络中其他神经元的感应局部区域。获胜神经元 i 的输出信号 y_i 被置为 1，所有竞争失败神经元的输出信号被置为 0，即

$$y_i = \begin{cases} 1, v_i > v_j & j \neq i \\ 0 & \text{其他} \end{cases} \tag{1-26}$$

感应局部场 v_k 表示神经元 k 的所有前向和反馈输入的组合行为。

令 w_{ij} 为输入 x_j 与某个神经元 i 的突触权值，假设分配给每个神经元固定数量的突触权重，即

$$\sum_j w_{ij} = 1 \qquad \text{对所有 } i \tag{1-27}$$

如果一个特定的神经元 i 在竞争中获胜，则这个神经元的每一个输入节点都放弃输入权值的一部分，并且放弃的权值平均分布在活性输入节点之中。根据标准竞争学习规则，突触权值的变化定义为

$$\Delta w_{ij} = \begin{cases} \eta(x_j - \Delta w_{ij}) & \text{如果神经元 } i \text{ 在竞争获胜} \\ 0 & \text{如果神经元 } i \text{ 在竞争失败} \end{cases} \tag{1-28}$$

式中，η 是学习速率参数。这个规则能够使得获胜神经元 i 的突触权值重向量 $\boldsymbol{w}_{ij}$ 向输入模式 $\boldsymbol{x}_j$ 转移。

5．随机学习规则

随机学习规则也称为 Boltzmann 学习规则，是为了纪念 Ludwig Boltzmann 而命名的。Boltzmann 学习规则是由统计力学思想而来的，在 Boltzmann 学习规则基础上设计出的神经网络称为 Boltzmann 机，其学习实质上就是著名的模拟退火（Simulated Annealing, SA）算法。

1.6　神经网络的特点及优点

1.6.1　神经网络特点

神经网络的主要特点是：

（1）分布式存储信息。其信息的存储分布在不同的位置，神经网络是用大量神经元之间的

连接及对各连接权值的分布来表示特定的信息，从而使网络在局部网络受损或输入信号因各种原因发生部分畸变时，仍然能够保证网络的正确输出，提高网络的容错性和鲁棒性。

（2）信息处理与存储合二为一。神经网络的每个神经元都兼有信息处理和存储功能，神经元之间连接强度的变化，既反映了对信息的记忆，同时又与神经元对激励的响应一起反映了对信息的处理。

（3）并行协同处理信息。神经网络的每个神经元都可根据接收到的信息进行独立的运算和处理，并输出结果，同一层中的各个神经元的输出结果可被同时计算出来，然后传输给下一层做进一步处理，这体现了神经网络并行运算的特点，这一特点使网络具有非常强的实时性。虽然单个神经元的结构极其简单，功能有限，但大量神经元构成的网络系统所能实现的功能是极其丰富多彩的。

（4）对信息的处理具有自组织、自学习的特点，便于联想、综合和推广。神经网络的神经元之间的连接强度用权值大小来表示，这种权值可以通过对训练样本的学习而不断变化，而且随着训练样本的增加和反复学习，这些神经元之间的连接强度会不断增加，从而提高神经元对这些样本特征的反映灵敏度。

1.6.2 神经网络的优点

神经网络的计算能力很明显有以下两个特点：一是大规模并行分布式结构；二是神经网络学习能力以及由此而来的泛化能力。泛化是指神经网络对不在训练集中的数据可以产生合理的输出。这两种信息处理能力让神经网络可以解决一些当前还不能处理的复杂的大型问题。

神经网络具有下列性质和能力：

（1）非线性。一个人工神经元可以是线性或者是非线性的。一个由非线性神经元互连而成的神经网络自身是非线性的，并且非线性是一种分布于整个网络中的特殊性质，且是一个很重要的性质。

（2）输入/输出映射。有监督学习或有导师学习是一个学习的流行范例，使用训练样本对神经网络的权值进行修改。每个样本由一个唯一的输入信号和相应的期望响应组成。从训练集中随机选取一个样本输入网络，网络就调整它的权值，使输入信号的期望响应与由输入信号通过网络计算而产生的实际响应之间的差别最小化。使用训练集中的很多样本重复网络的训练，直到网络到达没有显著的权值修正的稳定状态为止。对当前问题，网络通过建立输入/输出映射从样本中进行学习。

（3）自适应能力。神经网络嵌入了一个调整自身权值以适应外界变化的能力。特别是一个在特定运行环境下接受训练的神经网络，对环境条件不大的变化可以容易地进行重新训练。而且，当它在一个时变环境（即它的统计特性随时间变化）中运行时，网络权值可以设计成随时间变化。用于模式识别、信号处理和控制的神经网络与它的自适应能力耦合，就可以变成能进行自适应模式识别、自适应信号处理和自适应控制的有效工具。作为一个一般规则，在保证系统保持稳定时一个系统的自适应性越好，当要求在一个时变环境下运行时它的性能就越具鲁棒性。但自适应性不一定导致鲁棒性，实际可能相反。

（4）证据响应。在模式识别问题中，神经网络可以设计成既提供选择哪一个特定模式的信息，也提供决策的置信度信息。后者可以用来判断那些出现的过于模糊的模式。有了这些信息，网络的分类性能就会改善。

（5）背景的信息。神经网络的特定结构和激发状态代表知识。网络中每一个神经元潜在地都受网络中所有其他神经元全局活动的影响。因此，背景信息自然由一个神经网络处理。

（6）较好的容错性。通常神经网络总是包含大量的神经元和大量的连接关系，同时信息是分布式存储的，当一些神经元遭到破坏，整个系统仍能正常工作，因而具有高度的容错能力和坚韧性。另外，一个以硬件形式实现后的神经网络有天生容错的潜质或者鲁棒计算的功能，即它的性能在不利运行条件下逐渐下降。

（7）VLSI 实现。神经网络的大规模并行性使它具有快速处理某些任务的潜在能力。这一性能使得神经网络很适合用超大规模集成（Very Large Scale Integrated，VLSI）技术实现。

（8）分析和设计的一致性。基本上，神经网络作为信息处理器具有通用性，即涉及神经网络应用的所有领域都使用同样的记号。这种特征以不同的方式表现出来；不管形式如何，神经元在所有的神经网络中都代表一个相同的成分。这种共性使得在不同应用中的神经网络共享相同的理论和学习算法成为可能。模块化网络可以用模块的无缝集成来实现。

（9）神经生物类比。神经网络的设计是由人脑的类比引发的，人脑是一个容错的并行处理的活生生的例子，说明这种处理不仅在物理上可实现，而且还是快速高效的。神经生物学家将人工神经网络看做一个解释神经生物现象的研究工具。

1.7　神经网络的结构

为了简化讨论，将前面 1.3 节讨论的几种人工神经网络结构作一个统一的描述，如图 1-15 所示。其中圆圈中的 N 表示神经元，对于图 1-6 和图 1-7 所示的神经网络结构，N 代表激活函数、变换函数；而对于图 1-8 和图 1-9，N 则表示加法器或者乘法器。$\boldsymbol{s}$ 表示样条权函数向量，$\boldsymbol{w}$ 表示传统人工神经网络结构中的常数权向量，但是不包含阈值；而 $\tilde{\boldsymbol{w}}$ 表示传统人工神经网络结构中的常数权向量，包含阈值。

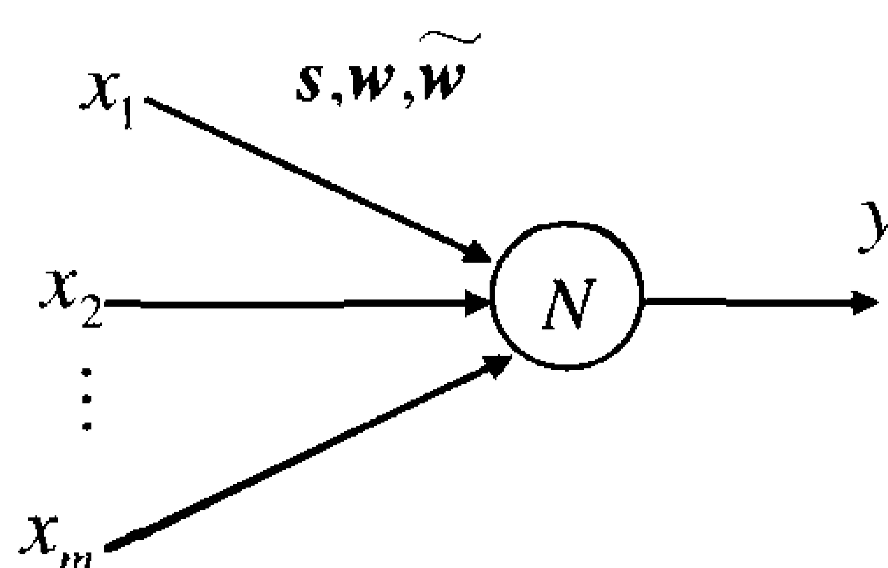


图 1-15　几种多输入单输出人工神经网络结构的统一描述

一般来说，有多个输入的单个神经元并不能满足实际应用的要求。在实际应用中需要有多个并行操作的神经元，这些可以并行操作的神经元组成的集合称为“层”。

图 1-16 中的小圆圈表示预处理单元（输入层和输出层都有预处理单元，预处理单元有时可以用来平滑数据），图 1-16 中的标有 N_i 的圆圈表示人工神经元，可以是样条函数神经元，也可以是传统神经元。x_i 表示人工神经网络的输入，y_i 表示人工神经网络的实际输出，z_i 表示人工神经网络的目标输出（即期望的输出）。神经网络训练的目的就是通过权的调整，使得 y_i 和 z_i 之间的误差最小。$\boldsymbol{S}$ 表示样条权函数矩阵。$\boldsymbol{W}$ 表示传统人工神经网络结构中的常数权值矩阵，但是不包含阈值，而 $\tilde{\boldsymbol{W}}$ 表示传统人工神经网络结构中的常数权矩阵，包含阈值。

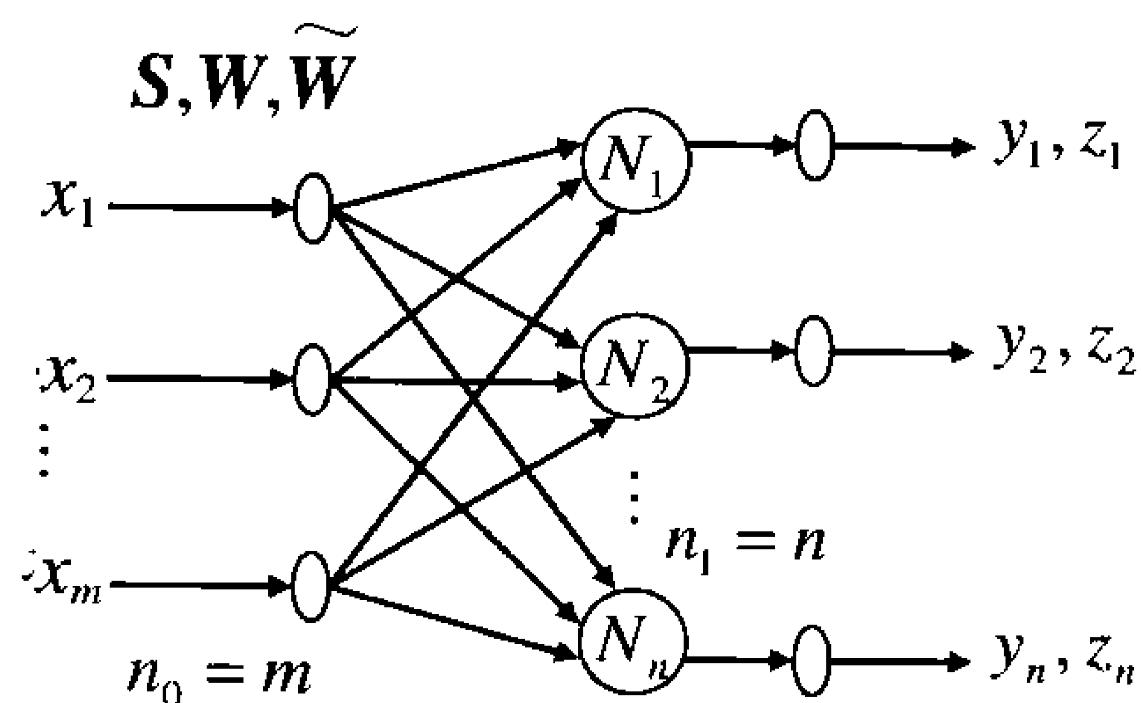


图 1-16　几种多输入多输出人工神经网络结构的统一描述

比较图 1-15 和图 1-16 可知，两者之间的区别仅仅是输出维数（即输出层节点个数）不同。

对于样条函数神经网络，图 1-16 所示的人工神经网络结构就可以实现 m 维输入到 n 维输出之间的映射，而对于代数算法，图 1-17 所示的人工神经网络结构可以实现 m 维输入到 n 维输出之间的映射，显然，这里需要更多的连接权。

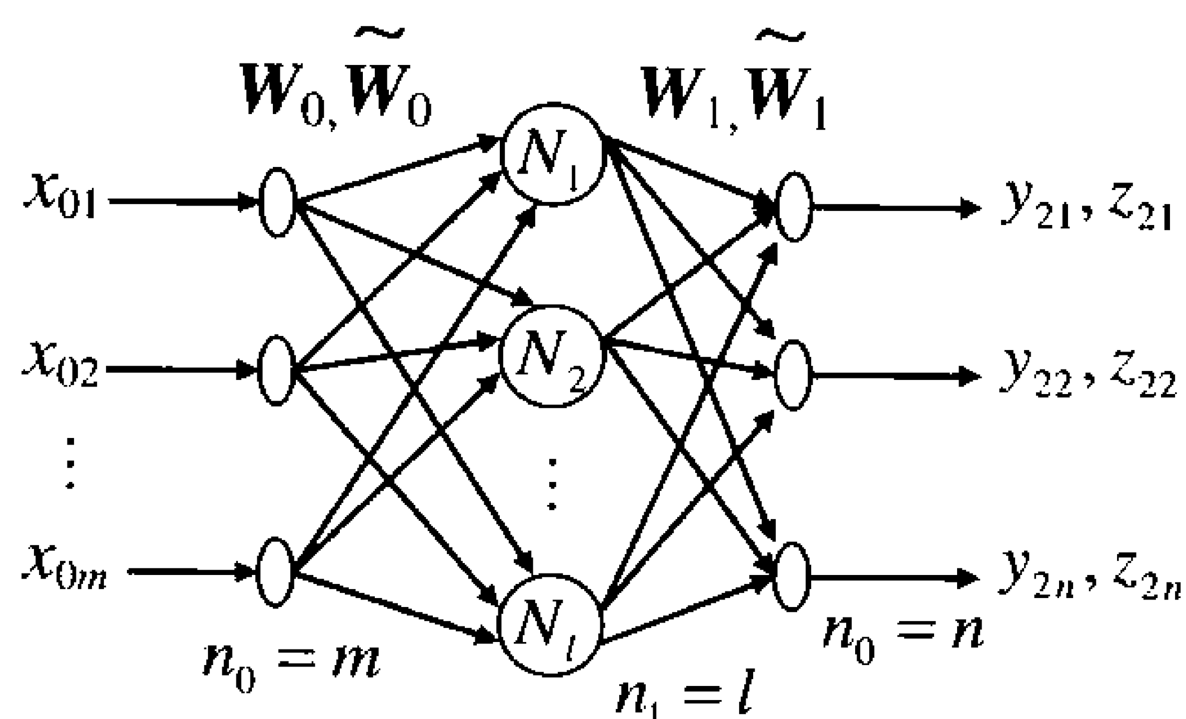


图 1-17　三层多输入多输出人工神经网络结构

有时为了描述方便，将神经网络进一步简化。图 1-18 是对于图 1-16 的简化。在图 1-18 中，$\boldsymbol{X}$ 表示输入矩阵，$\boldsymbol{Y}$ 表示输出矩阵，粗箭头上方的 $\boldsymbol{W}_{\mathrm{AC}}$ 表示全互连的前馈连接权，其含义是输入层的每一个神经元的输出都前馈连接到输出层的每一个神经元。

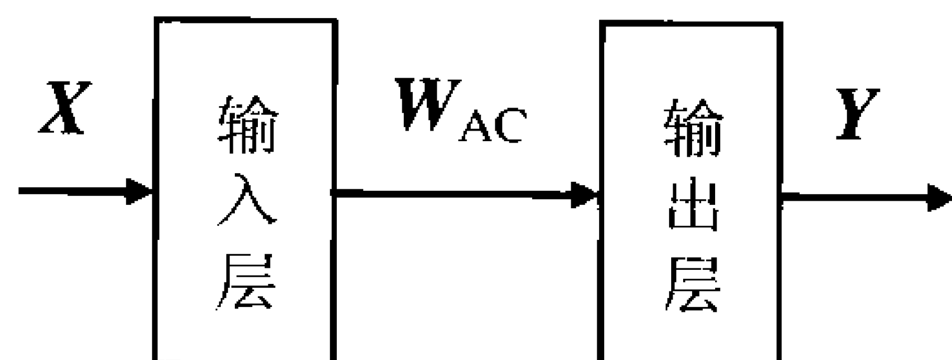


图 1-18　多输入多输出人工神经网络结构简化图示

图 1-19 是对于图 1-17 的简化。在图 1-19 中，$\boldsymbol{X}$ 表示输入矩阵，$\boldsymbol{Y}$ 表示输出矩阵，粗箭头上方的 $\boldsymbol{W}_{0\mathrm{AC}}$ 表示输入层与隐层之间的全互连的前馈连接权，其含义是输入层的每一个神经元的输出都前馈连接到隐层的每一个神经元。$\boldsymbol{W}_{1\mathrm{AC}}$ 则表示隐层与输出层之间的全互连的前馈连接权，其含义是隐层的每一个神经元的输出都前馈连接到输出层的每一个神经元。

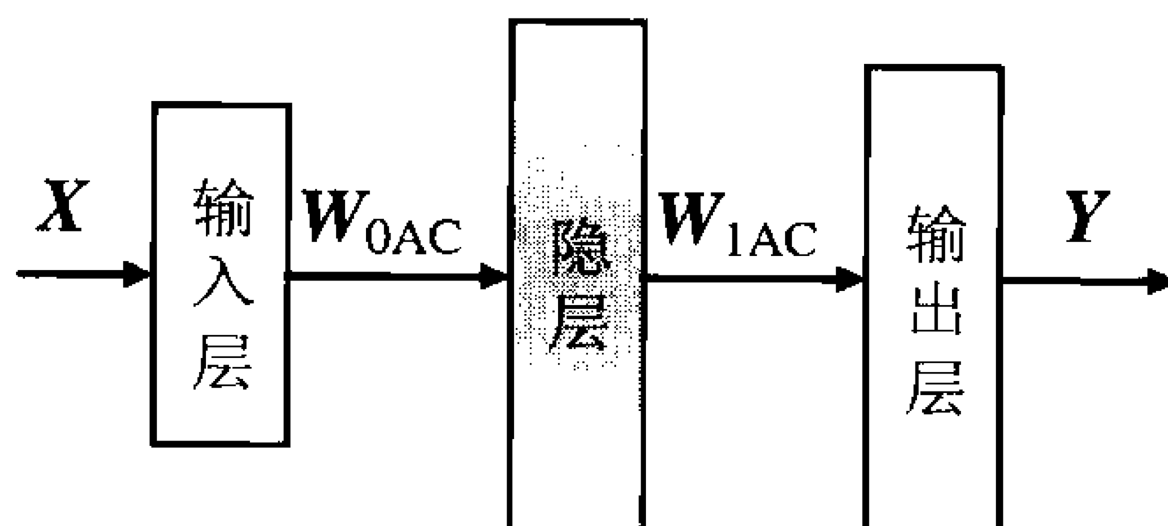


图 1-19　三层多输入多输出人工神经网络结构简化图示

以上简化图示的方法可以方便地推广到多个隐层时人工神经网络的结构，如图1-20所示。

在图1-20中，假设人工神经网络总共有L层，记为第$k=1,2,\cdots,L-1$层。第0层$(k=0)$为输入层，第$L-1$层（$k=L-1$）为输出层，其余中间各层是隐层。

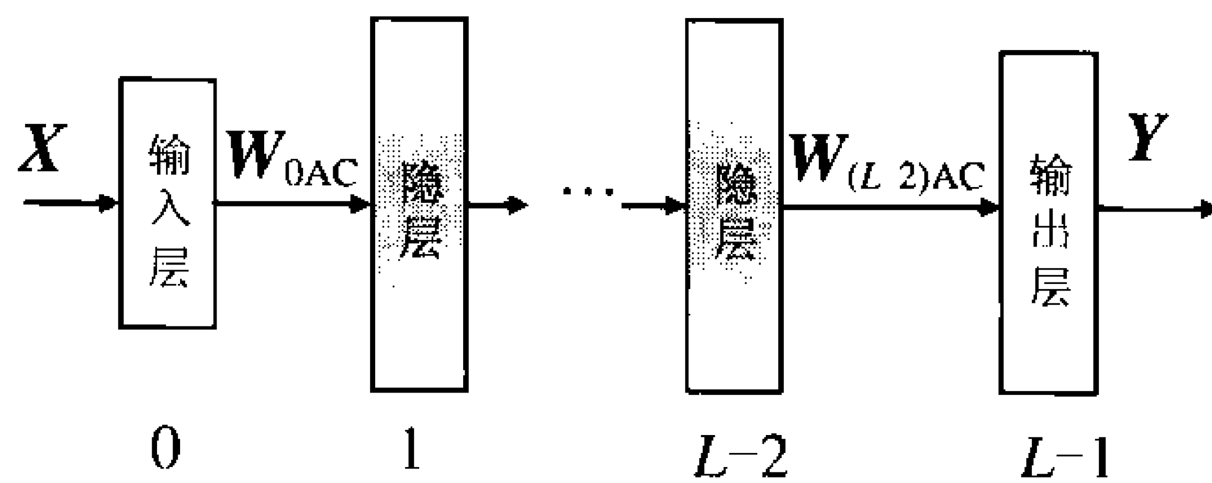


图1-20 多隐层多输入多输出人工神经网络结构的简化图示

神经网络的算法通常可以用矩阵运算来描述。全书将采用矩阵描述方法。

当然，神经网络的结构不止以上几种。如果各个层之间具有反馈连接，则构成了递归神经网络。

1.8 人工神经网络与人工智能

人工智能（Artificial Intelligence, AI）是用计算机模型拟思维功能的科学。人工智能与人工神经网络均是研究人脑、人类思维和认知的学科与技术，它们之间存在着千丝万缕的联系，本节将从认识模型上对人工智能与人工神经网络进行比较。

1.8.1 人工智能的概述

人工智能是计算机科学的一个重要分支，是跨学科的前沿科学，涉及心理学、脑生理学、计算机科学、哲学等学科。20世纪30至40年代，数量逻辑和关于计算的新思路这两项崭新的研究成果促成了人工智能的产生和发展。数量逻辑的研究成果使推理的某些方面可以用比较简单的结构予以形式化，从而为智能活动的部分过程在计算机上实现打下了基础。计算的新思路：图灵于1946年提出的“思维即计算”（“Thinking is computing”）把符号处理过程中的形式推理升到思维的高度。

对于人工智能有不同的定义。

广义人工智能的定义：通过对人类智能活动奥秘的探索与记忆思维机理的研究，以实现两方面的目的：（1）开发人类智力活动的潜能；（2）探讨用各种（电气的、光学的、生物的，甚至机械的）机器模拟人类智能的途径，使人类的智能得以物化与延伸。

狭义人工智能的定义：人工智能是用计算机模型模拟思维功能的科学。

人工智能必须有能力做三件事：知识存储、用存储的知识解决问题和通过经验获取新知识，故一个人工智能系统有三个关键部分：表示、推理和学习，如图1-21所示。

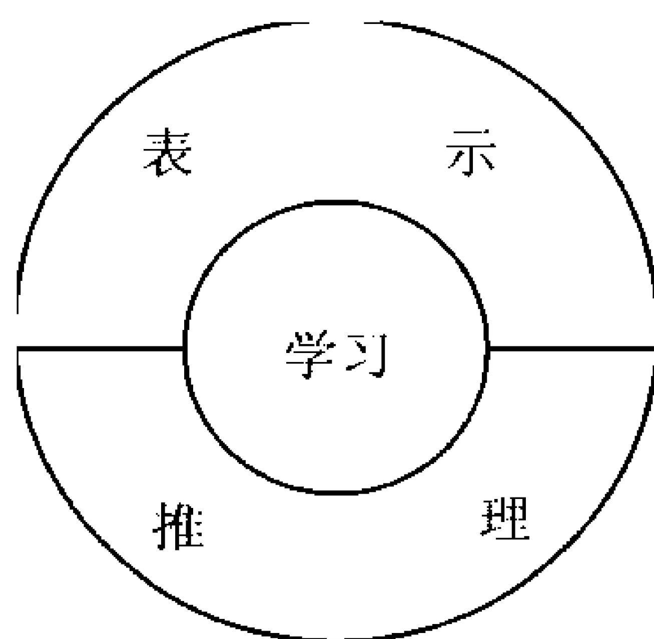


图1-21 人工智能系统的三个核心部分示意图

1．推理

在许多基本结构中，推理是解决问题的能力，一个系统要有出色的推理能力，必须要满足特定条件：①系统必须支持显示和隐藏的信息；②系统必须能表示和解决十分广泛的问题及问题类型；③系统必须有一个控制机制，当问题已被求解或对问题的进一步处理完成时，以决定对特定问题使用何种操作。寻找问题的解决可视为搜索，对“搜索”通常的处理方法是用规则、数据和控制。规则作用于数据，控制作用于规则。

在实际情况中（如医学诊断）可能遇到可用知识不完全或不确切的情况，在这种情况下，可以采用概率推理过程，因此人工智能系统可处理不确定性。

2．表示（Representation）

人工智能最独特的特性是对符号结构语言普遍深入的应用，这种语言能表示特定问题域的一般知识和问题求解的特殊知识，符号通常用公式表示，这种表示对用户而言相对容易理解，实际上符号人工智能的透明度非常适合人机交互。

人工智能研究专家用到的“知识”是数据的另一种表述，具有说明性和过程性。在说明性表示中，知识表示为事实的静态集合，并带有一个用于操作事实的一般性过程集合。在过程性表示中，知识表示蕴含于可执行代码中，此代码能执行知识表达的意义。在绝大部分问题域中通常用这两种类型的知识。

3．学习

学习也称为机器学习。机器学习的简单模型如图 1-22 所示。

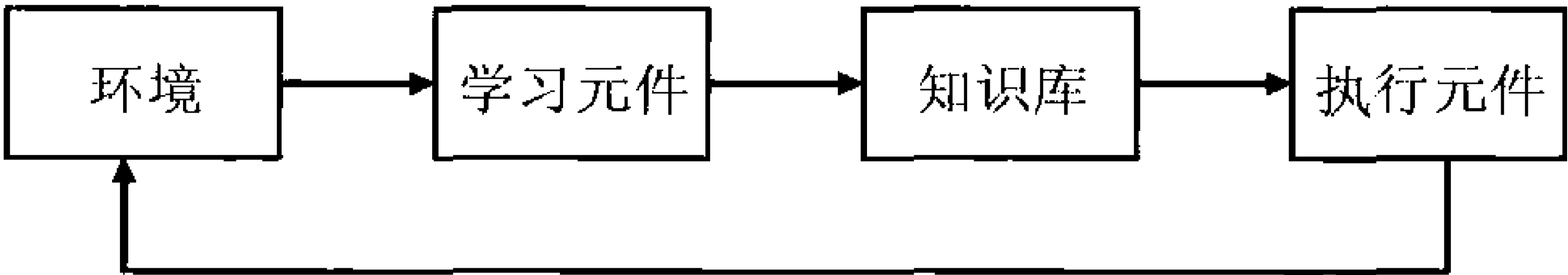


图 1-22　机器学习的简单模型

环境提供给学习元件一些信息，学习元件将这些信息加入到知识库中，执行元件以知识库为基础执行任务。周围环境提供给机器的知识种类通常是有缺陷的，结果学习元件事先不知道如何填补遗失细节或如何忽略不重要的细节，因此机器先凭猜测执行，再获取从执行元件传来的反馈，反馈机制使机器能推测假设并在必要时进行修正。

机器学习包括两种截然不同的信息处理方向：归纳和演绎。在归纳信息处理过程中，从原始数据和经验得到一般模式和规则。在演绎信息处理中，从一般规则得到特定事实。基于相似度的学习用归纳的方法，其理论证据则是从已知公理和理论中演绎而来的；基于解释的学习用归纳和演绎两种方法。

1.8.2　人工神经网络的应用

人工神经网络的应用极其广泛，1988 年《DARPA 神经网络研究报告》列举了神经网络在不同领域中的应用实例。

（1）航空航天业：高性能飞机自动驾驶仪、飞行航线模拟、飞行器控制系统、自动驾驶增强、飞机构件模拟、飞机构件故障检测等。

（2）汽车业：汽车自动驾驶系统、保单行为分析。

（3）银行业：支票和其他文档读取、信用卡申请书评估。

（4）信用卡行为检查：用于辨认与遗失的信用卡相关的不寻常的信用卡行为。

（5）国防工业：武器制造、目标跟踪与识别、脸部认识、新型传感器、声呐、雷达、图像处理与数据压缩、特征提取与噪声抑制、信号/图像识别。

（6）电子业：编码序列预测、集成电路芯片版图设计、过程控制芯片故障检测、机器人视觉、语音合成非线性建模。

（7）娱乐业：动画、特效、市场预测。

（8）金融业：房地产评价、贷款指导、抵押审查、集团债务评估、信用曲线分析、有价证券交易程序、集团财政分析、货币价格预测等。

（9）工业：预测熔炉产生的气体，在其他工业过程中取代复杂而昂贵的仪器设备。

（10）保险业：政策应用评估，产出最优化。

（11）制造业：制造业过程控制、产品设计与分析、过程与机器诊断、实时微粒识别、可视化质量检测系统、焊接质量分析、纸质预测、计算机芯片质量分析、化学产品设计分析、机器保养分析、工程投标、经营与管理、化学处理系统的动态建模等。

（12）医学：乳腺癌细胞分析、EEG 和 ECG 分析、假体设计、移植时间最优化。降低医疗费用支出，提高医疗质量。

（13）石油天然气：勘探。

（14）机器人技术：行走路线控制、铲车机器人、操纵控制器、视觉系统等。

（15）语音：语音识别、语音压缩、元音分类、文本-语音合成等。

（16）有价证券：市场分析、自动债券评级、股票交易咨询系统等。

（17）电信业：图像与数据压缩、自动信息服务、实时语言翻译、用户付费处理系统等。

（18）交通：卡车刹车诊断系统、车辆调度、行程安排系统等。

相信随着人工神经网络研究的进一步深入，特别是人工神经网络作为一种智能方法同其他学科领域更为紧密的结合，人工神经网络的应用前景将更为广阔。

1.8.3 人工神经网络与人工智能相比较

人工智能与人工神经网络在认知模型上许多不同之处，我们从解释级别、表示结构和处理方式 3 个方面进行比较。

1．解释级别

传统人工智能中用符号表示某些事物。从认知的观点来看，人工智能认为智能表示方法是存在的，并且建立了由符号表示的顺序处理的认知模型。

人工神经网络强调并行分布处理（Parallel Distributed Processing, PDP）模型，这种模型认为通过大量神经元的相互作用产生信息处理，每一个神经元在网络中将兴奋或抑制信号送给其他神经元。神经网络特别强调对认知现象的神经生物解释。

2．表示结构

符号表示含有一个准语言结构。同自然语言的表达一样，经典 AI 的表达通常非常复杂，它建立于简单符号的系统结构之上。给定一个有限符号集，有意义的新表达通常由下列部分组成：符号表达成分、句法结构与语义的模拟。

而结构和天然表示是神经网络的一个至关重要的问题，于 1988 年定期举行的认知会议上，Fodor 和 Pylyshyn 含蓄地批判了神经网络处理认知和语言方面的计算复杂度，他们就神经网络在认知的两个基本过程，即思维表示和思维处理中存在哪些误解产生了分歧。根据 Fodor 和 Pylyshyn 思想，提出了传统人工智能而非神经网络的理论：

- 思维表示的特征表现出一个复合结构的组成和组合语义。
- 思维过程对运行一个复合结构的组成和组合语义非常敏感。

3．处理方式

传统人工智能中，处理是顺序的，类似于典型的计算机编程，甚至当没有预定次序（如专家系统中对事实和规定的扫描）时，其处理操作仍然是一步一步执行。

相比之下，并行性不仅对神经网络中的信息处理至关重要，而且是适应性之源，庞大的并行度（成千上万神经元）赋予了神经网络鲁棒性这一重要特征。计算在众多网络的神经元中传播，网络对含有哭声或不完整的输入仍然可以进行识别；一个被损坏的网络仍然可以满足运行，而且其学习无须太好。

总之，我们可用自上而下的方法将符号人工智能描述为知识数据表示和规则系统语言中一个刻板机的操作，也可用自下而上的方法将神经网络描述为具有天然学习能力的并行分布式处理机。在执行认知任务的过程中，应发现更多隐含的有用的方法、手段并将其结合成为结构化的连接模型混合系统，而不是单独的基于符号的人工智能或神经网络进行求解搜索，因此，我们能将神经网络的适应性、鲁棒性和一致性与从符号人工智能中继承的表示、推理和普通性结合起来，产生期望特性。实际上，一些方法就是基于这种目标思想而产生的，能从训练的神经网络中提取规则，另外对符号和联机结合方式上理解，可以整合起来制造一个智能机。

第 2 章　MATLAB 语言及神经元

2.1　MATLAB 简介

MATLAB 诞生于 20 世纪 70 年代，它的编写者是 Cleve Moler 博士和他的同事。当时，他们利用 Fortran 开发了两个子程序库——EISPACK 和 LINPACK。这两个子程序库是求解线性方程的程序库。但是，Cleve Moler 发现学生使用这两个程序库有些困难，主要问题是因为接口程序不好写，很费时间。于是 Cleve Moler 自己动手，在业余时间里编写了 EISPACK 和 LINPACK 的接口程序。Cleve Moler 给这个接口程序取名为 MATLAB，意为矩阵（Matrix）和实验室（Laboratory）的组合。以后几年，MATLAB 作为免费软件在大学里被广泛使用，深受大学生的喜爱。

1984 年，Cleve Moler 和 John Little 成立了 MathWorks 公司，正式把 MATLAB 推向市场，并继续进行 MATLAB 的开发。1933 年，MathWorks 公司推出了 MATLAB 4.0；1995 年，MathWorks 公司推出了 MATLAB 4.2 C 版（For Win3.x）；1997 年，推出了 MATLAB 5.0；2000 年 10 月，MathWorks 公司推出了 MATLAB 6.0；2002 年 8 月，发布了 MATLAB 6.5；2004 年 9 月发布了 MATLAB 7；2006 年 3 月发布了 MATLAB 2006a；2006 年 9 月发布了 MATLAB 2006b；2007 年 3 月发布了最新的 MATLAB R2007。每一次版本的推出都使 MATLAB 有长足的进步，界面越来越友好，内容越来越丰富，功能越来越强大，帮助系统越来越完善。

MATLAB 善长数值计算，能处理大量的数据，而且效率比较高。MathWorks 公司在此基础上加强了 MATLAB 的符号计算、文字处理、可视化建模和实时控制能力，增强了 MATLAB 的市场竞争力，使 MATLAB 成为市场注流的数值计算软件。

MATLAB 产品族是支持从概念设计、算法开发、建模仿真到实时实现的理想的集成环境的。无论进行科学研究还是产品开发，MATLAB 产品族都是必须不可少的工具。

1．MATLAB 产品构成

MATLAB 主要产品有：

● MATLAB

所有 MathWorks 公司产品的数值分析和图形基础环境。MATLAB 将 2D 和 3D 图形、MATLAB 语言编程集成到一个单一的、易学易用的环境之中。

● MATLAB ToolBox

一系列专用的 MATLAB 函数库，用于解决特定领域的问题。工具箱是开放的、可扩展的，可以查看其中的算法或开发自己的算法。

● MATLAB Compiler

将 MATLAB 语言编写的 M 文件自动转换成 C 或 C++文件，支持用户进行独立的应用开发。结合 MathWorks 提供的 C/C++数学库函数和图像库，用户可以利用 MATLAB 快速地开发出功能强大的独立应用系统。

● Simulink

它结合了框图界面和交互仿真能力的非动态系统仿真工具。它以 MATLAB 的数学、图形

和语言为基础。

- Stateflow

与 Simulink 框图模型相结合，描述复杂事件驱动系统的逻辑行为，驱动系统可以在不同的模式之间进行切换。

- Real-Time Workshop

直接从 Simulink 框图自动生成 C 或 Ada 代码，用于实现快速原型和硬件的仿真，整个代码的生成可以根据需要完全定制。

- Simulink Blockset

专门为特定领域设计的 Simulink 功能模块的集合，用户也可以利用已有的模块或自行编写 C 和 MATLAB 程序建立自己的模块。

2. MATLAB 的功能

MATLAB 的核心是一个基于矩阵运算的快速解释程序，它交互式地接收用户输入的各项命令，输出计算结构。MATLAB 提供了一个开放式的集成环境，用户可以运行系统提供的大量命令，包括数据计算、图形绘制和代码编制等。具体来说，MATLAB 具有以下功能。

- 矩阵运算功能
- 数据可视化功能
- 绘图功能
- 大量的工具箱
- GUI 设计
- Simulink 仿真

通过运用 MATLAB 这些强大的功能，工程师、科研人员、数学家和教育工作者可以在统一的平台下完成相应的科学计算工作。这些工具涉及的领域非常广泛，例如汽车、电子仪器表和通信等。

MathWorks 公司刚刚推出了 MATLAB R2007 版本产品，即 MATLAB R2007，增加了两个新产品模块，同时还升级和修正了 82 个产品模块。

3. MATLAB 的应用领域

MATLAB 产品族可用于以下领域。

- 数据分析
- 数值和符号计算
- 建模、仿真、原型开发
- 控制系统设计
- 工程与科学绘图
- 图形用户界面设计等
- 数据图像信号处理
- 财务工作

MATLAB 产品族被广泛地应用于包括信号与图像处理、控制系统设计、系统仿真等诸多领域。开放式的结构使 MATLAB 产品族很容易针对特定的需求进行扩充，从而在不断深化对问题的认识的同时，提高自身的竞争力。

MATLAB 产品族的一大特性是有众多的面向具体应用的工具箱和仿真模块，包含了完整的函数集，用来对信号与图像处理、控制系统设计、神经网络等特殊应用进行分析和设计。同时，

其他的产品也延伸了 MATLAB 的能力，包括数据采集、报告生成和依靠 MATLAB 语言编程产生独立的 C/C++代码等。

2.2　MATLAB 的语言特点

MATLAB 语言具有不同于其他高级语言的特点，被称为“第四代”计算机语言。与第三代计算机语言如 FORTRAN 语言和 C 语言一样，MATLAB 语言使人们从烦琐的程序代码中解放出来。其丰富的函数使开发者无须重复编程，只要简单地调用和使用即可。下面将一一介绍 MATLAB 语言的主要特点。

1．编程效率高

MATLAB 是一种面向科学与工程计算的高级语言，允许用数学形式的语言编写程序，且比 FORTRAN 和 C 语言更加接近书写计算公式的思维方式，用 MATLAB 语言编程犹如在演算纸上排列出公式和求解问题。因此，MATLAB 语言也可通俗地称为“演算纸式”科学算法语言。由于它编写简单，所以编程效率高，易学易懂。

2．扩充能力好，交互性好

高版本的 MATLAB 语言具有丰富的库函数，在进行复杂的数学运算时可以直接调用，而且 MATLAB 的库函数与用户文件在形式上一样，所以用户文件也可以作为 MATLAB 的库函数来调用。因而，用户可以根据自己的需要方便地建立和扩充新的库函数，以便提高 MATLAB 的使用效率。另外，它还充分利用 FORTRAN、C 等语言的资源，包括用户已编好的 FORTRAN、C 语言程序，方便地调用有关 FORTRAN、C 语言的子程序，还可以在 C 语言和 FORTRAN 语言中方便地使用 MATLAB 的数值计算功能，这样良好的交互性使程序员可以使用以前编写过的程序，以减少重复性工作，也使现在编写的程序具有重复利用的价值。

3．移植性和开放性好

MATLAB 是用 C 语言编写的，且 C 语言的可移植性很好，因此 MATLAB 可以很方便地移植到可运行 C 语言的操作平台上。MATLAB 适合的工作平台有 Windows 系列、UNIX、Linux、PowerMac、VMS6.1。除了内部函数外，MATLAB 所有的核心文件和工具箱文件都是公开的，都是可读可写的源文件，用户可以通过对源文件的修改和自己编程建立新的工具箱。

4．语句简单、内涵丰富

MATLAB 语言中最基本的成分是函数，函数一般由变量名、输入变量和输出变量组成。同一个函数名，不同数目的输入变量及不同数目的输出变量代表着不同的含义。这不仅使 MATLAB 的库函数功能更丰富，而且大大减小了需要的硬盘空间，使得 MATLAB 编写的 M 文件简单、短小而高效率。

5．用户使用方便

人们用任何一种语言编写程序和调试程序时一般都要经过四个步骤：编辑、编译、连接及执行和调试。各个步骤之间是顺序关系，编程的过程就是在它们之间作瀑布型的循环。MATLAB 语言与其他语言相比，较好地解决了上述问题，将编辑、编译、连接和执行融为一体。它能在同一画面上进行灵活操作，快速排除输入程序中的书写错误、语法错误及语义错误，从而加快了用户编写、修改和调试程序的速度，可以说在编程和调试过程中它是一种比 VB 还要简单的语言。

具体地说，MATLAB 运行时，如直接在命令行输入 MATLAB 语句，包括调用 M 文件的语句，就立即对其进行处理，完成编译、连接和运行的全过程。又如，将 MATLAB 源程序编辑

为 M 文件，由于 MATLAB 磁盘文件也是 M 文件，所以编辑后的原文件就可直接运行，而不需进行编译和连接。在运行 M 文件时，如果有错，计算机屏幕上会给出详细的出错信息，用户修改后再执行，直到正确为止。可以说，MATLAB 语言不仅是一种语言，而且是一种语言开发系统，即语言调试系统。

6．高效方便的矩阵和数组运算

与 BASIC、FORTRAN、C 语言一样，MATLAB 语言规定了矩阵的算术运算符号、关系运算符号、逻辑运算符号、条件运算符号及赋值运算符号，而且这些运算符号大部分都可以应用于数值间的运算。另外，它不需要定义数组的维数，而且可提供矩阵函数、特殊矩阵专门的库函数，使之在求解诸如信号处理、建模、系统识别、控制、优化等领域的问题时，显得大为简捷、高效、方便，这是其他高级语言不能比拟的。在此基础上，高版本的 MATLAB 已逐步扩展到科学及工程计算的其他领域中了。因此，不久的将来，它绝对有可能名副其实地成为“万能演算式”科学算法语言。

7．方便的绘图功能

图形和可视化是当代应用软件发展的主要方向。随着 MATLAB 版本的不断提高，MATLAB 的图形功能也越来越强。人们很难从一大堆原始离散数据中感受到它们的含义，但数据图形恰使人们直接感受到数据的许多内在本质。因此，数据可视化是人们研究科学、认识世界不可缺少的手段。

MATLAB 的绘图十分方便，它有一系列绘图函数，例如线性坐标、对数坐标、半对数坐标及极坐标。用户只需调用不同的绘图函数，在图上标出图题，*X*、*Y* 轴标注、格（栅）绘制也只需调用相应的命令，简单易行。另外，在调用绘图函数时，调整自变量可给出不同颜色的点、线、复线或多重线。

2.3 MATLAB 7.2 的新特点

MATLAB 7.2（2006a）于 3 月 1 日正式发布，并在 3 月 3 日开始对客户出货。在 MATLAB R2006a 中（MATLAB 7.2，Simulink 6.4），主要更新了 10 个产品模块、增加了多达 350 个新特性、增加了对 64 位 Windows 的支持，并新推出了.net 工具箱。

作为和 Mathematica、Maple 并列的三大数学软件，其强项就是其强大的矩阵计算及仿真能力。要知道 MATLAB 的由来就是 Matrix+Laboratory=MATLAB，所以这个软件在国内也被称做“矩阵实验室”。每次 MathWorks 发布 MATLAB 的同时也会发布仿真工具 Simulink。在欧美，很多大公司在将产品投入实际使用之前都会进行仿真试验，他们主要使用的仿真软件就是 Simulink。MATLAB 提供了自己的编译器：全面兼容 C++及 FORTRAN 两大语言。所以 MATLAB 是工程师、科研工作者手上最好的语言、工具和环境。MATLAB 已经成为广大科研人员最值得依赖的助手和朋友。

MATLAB 7.2 这次的升级做了重大的增强，也升级了以下各版本，提供了 MATLAB、Simulink 的升级以及其他最新模块的升级。MATLAB 7.2 版本不仅仅提高了产品质量，同时也提供了用于数据分析、大规模建模、固定点开发、编码等新特征。

MATLAB 7.2 主要更新的产品模块为：

- Control System Toolbox 7
- Embedded Target for T1 C2000(tm) DSP 2

- Embedded Target for T1 C6000(tm) DSP 2
- Financial Toolbox 3
- Link for ModelSim(r) 2
- MATLAB Report Generator 3
- Neural Network Toolbox 5
- Simulink Report Generator 3
- Simulink Report Optimization 3

MATLAB 7.2 新版本中，推出了下面三个新产品：

- MATLAB Builder for .net
- SimBiology(R14 SP3)
- SimEvents(R14 SP3)

其中 MATLAB Builder for .net 扩展了 MATLAB Compiler 的功能，主要有：

（1）可以打包 MATLAB 函数，使网络程序员可以通过 C、VB、.net 等语言访问这些函数。

（2）创建组件来操持 MATLAB 的灵活性。

（3）创建 COM 组件。

（4）将源自 MATLAB 函数的错误作为一个标准的管理异常来处理。

MATLAB 7.2 版本中，产品模块进行了一些调整，MATLAB Builder for COM 的功能集成到 MATLAB Builder for .net 中，Financial Time Series Toolbox 的功能集成到 Financial Toolbox 中。MATLAB 将高性能的数值计算和可视化集成在一起，并提供了大量的内置函数，从而被广泛地应用于科学计算、控制系统、信息处理等领域的分析、仿真和设计工作中，而且利用 MATLAB 产品的开放式结构，可以非常容易地对 MATLAB 的功能进行扩充，从而在不断深化对问题认识的同时，不断完善 MATLAB 产品，以提高产品自身的竞争能力。

目前 MATLAB 产品族可以用来进行：

- 数值分析
- 数值和符号计算
- 工程与科学绘图
- 控制系统的设计和仿真
- 数字图像处理
- 数字信号处理
- 通信系统设计与仿真
- 财务与金融工程

2.4　MATLAB 神经网络工具箱

2.4.1　MATLAB 6.x 神经网络工具箱

MATLAB 神经网络工具箱 4.0 版本是 MathWorks 公司最新推出的 MATLAB 6.x 高性能可视化数值计算软件的组成部分。它主要针对神经网络系统的分析与设计，提供了大量可供直接调用的工具箱函数、图形用户界面和 Simulink 仿真工具，是进行神经网络系统分析与设计的绝佳工具。与以往的 MATLAB 神经网络工具箱相比，神经网络工具箱 4.0 版本出现了许多新增功能，使用起来更加方便。这些新增功能主要包括：

- 新增了三个专门用于基本神经网络控制系统应用与设计的 Simulink 模块：神经网络预测控制器模块、反馈线性化控制器模块和模型参考自适应控制器模块，极大地方便了基于神经网络的控制系统设计与仿真。
- 改进并新增了四个神经网络训练函数：trainb、trainc、trainr、trains，其中 trainb 用于批量式训练，其他三个函数则用于渐进式训练。
- 提供了神经网络设计与仿真 CUI，使用户能够方便地通过图形用户界面进行神经网络的设计与仿真。
- 改进的"提前停止"算法可以和叶斯正则化方法联合使用，从而可以更好地提高神经网络的推广或泛化能力。
- 改进了线性神经网络设计函数 newlind，使其能够对多输入、多输出的线性神经网络进行直接设计。

2.4.2 MATLAB 7.x 神经网络工具箱

快速发展的 MATLAB 软件为神经网络理论的实现提供了一种便利的仿真手段。MATLAB 神经网络工具箱的出现，更加拓宽了神经网络的应用空间。神经网络工具箱将很多原本需要手动计算的工作交给计算机，一方面提高了工作效率；另一方面还提高了计算的准确度和精度，减轻了工程人员的负担。

MATLAB R2007 对应的神经网络工具箱的版本号为 Version 5.0.2，它以神经网络理论为基础，利用 MATLAB 脚本语言构造出典型神经网络的激活函数，如线性、竞争性和饱和线性等激活函数，使设计者对所选定网络输出的计算，变成对激活函数的调用。另外，根据各种典型的修正网络权值的规则，再加上网络的训练过程，利用 MATLAB 编写出各种网络设计和训练的子程序，网络设计人员可以根据自己的需要去调用工具箱中有关的设计和训练程序，将自己从烦琐的编程中解脱出来，集中精力解决其他问题，从而提高了工作效率。

最新版本的神经网络工具箱几乎涵盖了所有的神经网络的基本常用模型，如感知器和 BP 网络等。对于各种不同的网络模型，神经网络工具集成了多种学习算法，为用户提供了极大的方便。另外，该工具箱中还包括了大量的演示实例，有助于初始学者加深理解。它所介绍的网络模型和实例可以通过 MATLAB 的帮助进行学习，前题是英语水平要足够高。如在 MATLAB 的命令窗口中输入 help nnet，即可得到神经网络工具箱的有关版本信息及函数列表，如：

```
>> help nnet
  Neural Network Toolbox
  Version 4.0.6 (R14SP3) 26-Jul-2005
```

以下为函数列表：

```
Graphical user interface functions.
    nntool      - Neural Network Toolbox graphical user interface.

  Analysis functions.
    errsurf   - Error surface of single input neuron.
    maxlinlr - Maximum learning rate for a linear layer.
……
```

限于篇幅，在此只介绍了一些节选部分。

目前，神经网络工具箱中提供的神经网络的模型类型主要有：

- 感知器网络
- BP 网络
- 线性网络
- 径向基函数网络
- 动态网络
- 自组织映射和向量量子化网络

目前，神经网络工具箱中提供的神经网络模型主要应用于：

- 信息处理和预测
- 函数逼近和模型拟合
- 故障诊断
- 神经网络控制

在实际应用中，面对一个具体的问题时，首先需要分析利用神经网络求解问题的性质，然后依据问题特点，提取训练和测试数据样本，确定网络模型。最后通过对网络进行训练、仿真等，检验网络的性能是否满足要求。下面介绍具体的过程。

1）网络模型的确定

这主要是如何根据问题的实际情况，选择模型的类型、结构等。另外，还可在典型网络模型的基础上，结合问题的具体情况，对原网络进行变形、扩充等，同时还可以采用多种网络模型的组合形式。

2）确定信息表达方式

将领域问题及其相应的领域知识转换为网络可以接受并处理的形式，即将领域问题抽象为适合于网络求解所能接受的某种数据形式。尽管在实际应用中，问题的形式会是多种多样的，但不外乎以下几种情况：

- 数据样本已知
- 数据样本之间相互关系不明确
- 数据样本的预处理
- 将数据样本分为训练样本和测试样本
- 输入/输出模式为连续的或者离散的
- 输入数据按照模式进行分类，模式可能会具有平移、旋转或伸缩等变化形式。

3）训练模式的确定

确定训练模式包括选择合理的训练算法、确定合适的训练步数、指定适当的训练目标误差，以获得较好的网络性能。

4）网络参数的选择

确定网络输入/输出神经元的数目，如果还是多层网络，还需要进一步确定隐含层神经元的个数。对于反馈神经网络，如 Hopfield 网络和 Elman 网络，还需要进一步地设置反馈神经元的有关属性。

5）网络测试

选择合理的测试样本，对网络进行测试，或者将网络应用于实际问题，检验网络性能，值得指出的是，网络测试过程需要遵循“交叉测试”的原则，保证测试的有效性、准确性和全面性。

下面用一个经典的多层前馈网络解决 XOR 问题的例子对上面的过程做一个简单的说明。

1）输入和输出数据

```
>> p=[0 0 1 1;0 1 0 1]
p =
     0     0     1     1
     0     1     0     1
>> t=[1,0,0,1]
t =
     1     0     0     1
```

2）网络模型的确定

```
>> net=newff(minmax(p),[2,1],{'tansig','purelin'},'trainlm'),
```

上面是一个建立多层前馈网络的命令，[2，1]分别为隐藏层和输出层的神经元数，tansig、purelin 分别为隐藏层和输出层的激发函数，trainlm 为一种训练算法。

3）网络训练和测试

```
[net,tr]=train(net,p,t)                          %网络训练
TRAINLM, Epoch 0/100, MSE 1.70784/0, Gradient 6.69358/1e-010
TRAINLM, Epoch 5/100, MSE 7.70372e-032/0, Gradient 5.58671e-016/1e-010
TRAINLM, Minimum gradient reached, performance goal was not met.
>>a=sim(net,p)                                   %用训练好的 net 网络仿真原来的输入矩阵 p
a =
    1.0000    0.0000         0    1.0000         %仿真的输出
```

从上面的例子我们可以看出，利用 MATLAB 神经网络工具盒，几个简单的命令就可以解决一些复杂的问题。

总之，神经网络工具箱是一个内容全面、操作方便的软件包，对于广大神经网络系统的研发者来说，掌握神经网络工具箱的应用将使得自己在工作中如虎添翼。

2.5 MATLAB 神经网络工具箱的对象与属性

为了便于读者理解和应用 MATLAB 中的神经网络工具箱函数，首先介绍 MATLAB 中定义的神经网络对象及其属性。

在 MATLAB 中把定义的神经网络看做一个对象，对象还包括一些子对象：输入向量、网络层向量、输出向量、目标向量、权值向量和阈值向量等，这样网络对象和各子对象的属性共同确定了神经网络对象的特性。网络属性除了只读属性外，均可以按照约定的格式和属性值的类型进行设置、修改、引用等，只读属性只能引用。引用格式为

网络名. [子对象]. 属性

例如，net.numInputs, net.biasConnect(1), net.inputConnect(1,2), net.inputs{1}.range。

为了说明网络对象、子对象及其属性，首先建立如图 2-1 和图 2-2 所示的神经网络 net1 和 net2。

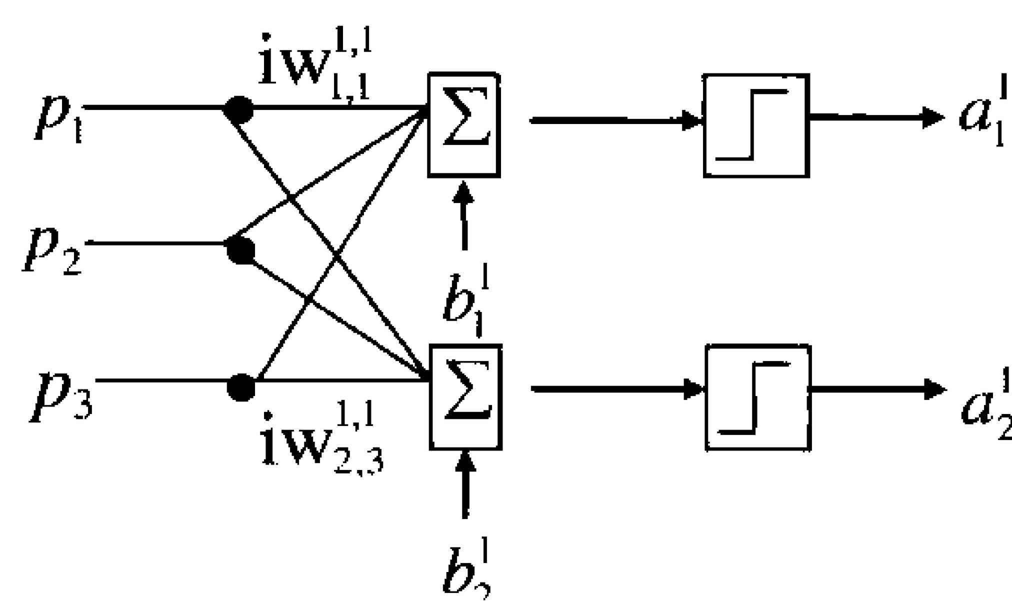


图 2-1　net1 的网络模型

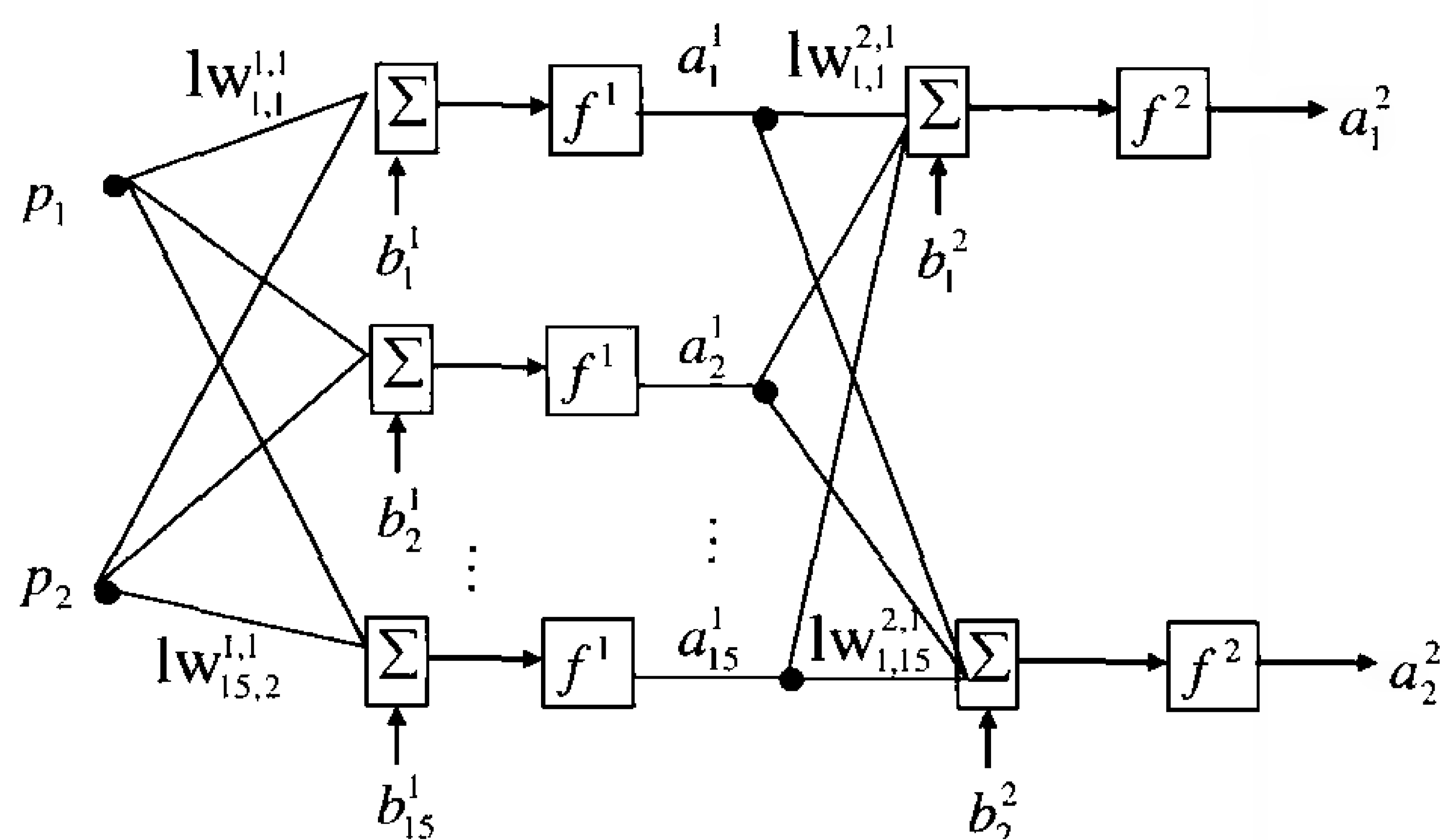


图 2-2　net2 的网络模型

在 MATLAB 命令窗口中逐条执行以下语句，即可创建网络 net1 和 net2。

```
>> p=[1 2;-1 1;0 1]
>> net1=newp(p,2)
net1 =
    Neural Network object:
    architecture:
          numInputs: 1
          numLayers: 1
        biasConnect: [1]
       inputConnect: [1]
       layerConnect: [0]
      outputConnect: [1]
      targetConnect: [1]
         numOutputs: 1    (read-only)
         numTargets: 1    (read-only)
    numInputDelays: 0    (read-only)
    numLayerDelays: 0    (read-only)
    subobject structures:
              inputs: {1x1 cell} of inputs
              layers: {1x1 cell} of layers
             outputs: {1x1 cell} containing 1 output
             targets: {1x1 cell} containing 1 target
              biases: {1x1 cell} containing 1 bias
       inputWeights: {1x1 cell} containing 1 input weight
       layerWeights: {1x1 cell} containing no layer weights
    functions:
            adaptFcn: 'trains'
             initFcn: 'initlay'
          performFcn: 'mae'
            trainFcn: 'trainc'
    parameters:
          adaptParam: .passes
           initParam: (none)
        performParam: (none)
          trainParam: .epochs, .goal, .show, .time
    weight and bias values:
                  IW: {1x1 cell} containing 1 input weight matrix
                  LW: {1x1 cell} containing no layer weight matrices
                   b: {1x1 cell} containing 1 bias vector
    other:
            userdata: (user stuff)
```

```
>> net2=newff([-1 1;-1 1],[15 2],{'tansig' 'purelin'},'traingdx','learngdm')
net2 =
    Neural Network object:
    architecture:
             numInputs: 1
             numLayers: 2
           biasConnect: [1; 1]
          inputConnect: [1; 0]
          layerConnect: [0 0; 1 0]
         outputConnect: [0 1]
         targetConnect: [0 1]
            numOutputs: 1    (read-only)
            numTargets: 1    (read-only)
        numInputDelays: 0    (read-only)
        numLayerDelays: 0    (read-only)
    subobject structures:
                inputs: {1x1 cell} of inputs
                layers: {2x1 cell} of layers
               outputs: {1x2 cell} containing 1 output
               targets: {1x2 cell} containing 1 target
                biases: {2x1 cell} containing 2 biases
          inputWeights: {2x1 cell} containing 1 input weight
          layerWeights: {2x2 cell} containing 1 layer weight
    functions:
              adaptFcn: 'trains'
               initFcn: 'initlay'
            performFcn: 'mse'
              trainFcn: 'traingdx'
    parameters:
            adaptParam: .passes
             initParam: (none)
          performParam: (none)
            trainParam: .epochs, .goal, .lr, .lr_dec,
                        .lr_inc, .max_fail, .max_perf_inc, .mc,
                        .min_grad, .show, .time
    weight and bias values:
                    IW: {2x1 cell} containing 1 input weight matrix
                    LW: {2x2 cell} containing 1 layer weight matrix
                     b: {2x1 cell} containing 2 bias vectors
    other:
              userdata: (user stuff)
```

2.5.1 网络对象属性

从上面的创建函数 newp 和 newff 创建的 net1 和 net2 后的显示结果可以看出所定义的神经网络 net1 和 net2 的属性。

1．结构属性

结构属性决定了网络子对象的数目（包括输入向量、网络层向量、输出向量、目标向量、权值向量和阈值向量的数目）以及它们的连接关系。无论何时，结构属性值一旦发生变化，网络就会自动重新定义，与之相关的其他属性值也会自动更新。

1）numInputs 属性

net.numInputs 属性定义了网络的输入向量数，它可以被设置为 0 或正整数。其值一般在用户自定义网络中才被设置，而由 MATLAB 神经网络工具箱中的网络定义函数创建的网络往往

是单个网络，其输入向量只有一个。如果用户定义了一个由多个网络构成的复杂的网络，其输入向量就不止一个，而是多个；而每个输入向量也可能包含了多个元素，其元素的个数由输入向量的大小（net.inputs{i}.size）决定，所以网络的输入向量数并不是网络输入元素（变量）的个数，这一点希望引起读者的注意。

2）numLayers 属性

net.numLayers 属性定义了网络的层数，它可以被设置为 0 或正整数。

例如上节显示结果中，net1.numLayers=1 表示 net1 只有一个网络层；net2.numLayers=2 表示 net2 有两个网络层。

net.numLayers 属性值一旦改变，下列与网络层相关的布尔代数矩阵的大小都会随之改变：

net.biasConnect

net.inputConnect

net.layerConnect

net.outputConnect

net.targetConnect

下列与网络层相关的子对象细胞矩阵的大小也会随之改变：

net.biases

net.inputWeights

net.layerWeigths

net.outputs

net.targets

下列网络调整参数细胞矩阵的大小也会随之改变：

net.IW

net.LW

net.b

上面提到的细胞矩阵（Cell Array）的概念可能对有的读者比较陌生，其实很简单，它将多个矩阵向量作为细胞矩阵的一个“细胞”（cell ），细胞矩阵的各个元素值为对应细胞的大小和数值类型，为区别于矩阵，用{}表示细胞矩阵。例如，设矩阵

$$\boldsymbol{a}=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix},\boldsymbol{b}=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix},\boldsymbol{c}=\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}$$

若将 $\boldsymbol{a}$、$\boldsymbol{b}$、$\boldsymbol{c}$ 分别作为细胞矩阵的一个细胞，则可以定义细胞矩阵 $\boldsymbol{m}$、$\boldsymbol{n}$ 为

$$\boldsymbol{m}=\begin{Bmatrix}\boldsymbol{a}\\ \boldsymbol{b}\end{Bmatrix}=\begin{Bmatrix}2\times 2\ \text{double}\\ 2\times 3\ \text{double}\end{Bmatrix},\boldsymbol{n}=\begin{Bmatrix}\boldsymbol{a} & []\\ \boldsymbol{b} & \boldsymbol{c}\end{Bmatrix}=\begin{Bmatrix}[2\times 2\ \text{double}] & []\\ [2\times 3\ \text{double}] & [3\times 2\ \text{double}]\end{Bmatrix}$$

可以通过下标，访问细胞矩阵中各细胞的值：

$$\boldsymbol{m}\{1\}=\boldsymbol{n}\{1,1\}=\boldsymbol{a}=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix},\boldsymbol{m}\{2\}=\boldsymbol{n}\{2,1\}=\boldsymbol{b}=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}$$

$$n\{2,2\}=c=\begin{bmatrix}1&2\\3&4\\5&6\end{bmatrix},n\{1,2\}=[]$$

3）biasConnect 属性

net.biasConnect 属性定义各个网络层是否具有阈值向量，其值为 $N_l\times1$ 布尔型向量(0 或 1)，N_l 为网络层数（net.numLayers）。

可以通过访问 net.biasConnect{i}的值，查看第 i 个网络层是否具有阈值向量。

net.baisConnect 属性值一旦改变，则阈值结构细胞矩阵（net.biases）和阈值向量细胞矩阵（net.b）将随之改变。

4）inputConnect 属性

net.inputConnect 属性定义各网络层是否具有来自各输入向量的连接权，其值为 $N_l\times N_i$ 的布尔型向量（0 或 1），N_l 为网络层数（net.numLayers），N_i 为网络输入向量数（net.numInput）。

可以通过访问 net.inputConnect(i,j)的值，查看第 i 个网络层是否具有来自第 j 个输入向量的连接权。

net.inputConnect 属性值一旦改变，输入层权值结构细胞矩阵（net.inputWeights）和权值向量细胞矩阵（net.IW）将随之改变。

5）layerConnect 属性

net.layerConnect 属性定义一个网络层是否具有来自另外一个网络层的连接权，其值为 $N_l\times N_l$ 的布尔型向量（0 或 1），N_l 为网络层数（net.numLaypers）。

可以通过访问 net.layerConnect(i,j)的值，查看第 i 个网络层是否具有来自第 j 个网络层的连接权。例如，在上节显示结果中，net1.layperConnect=[0]和 net2.layerConnect=[0 0; 1 0]，说明 net1 唯一的一个网络层没有来自本层的连接权（即无层内连接）；net2 的第二个网络层具有来自第一个网络层的连接权，而第一个网络层没有来自第二个网络层的连接权（即无反馈），各网络层也没有来自本层的连接权（无层内连接）。

net.layerConnect 属性值一旦改变，网络层权值结构细胞矩阵（net.layerWeights）和网络层权值向量细胞矩阵（net.IW）将随之改变。

6）outputConnect 属性

net.outputConnect 属性定义各网络层是否作为输出层，其值为 $1\times N_l$ 的布尔型向量(0 或 1)，N_l 为网络层数（net.numLayers）。

可以通过访问 net.outputConnect(i)的值，查看第 i 个网络层是否作为输出层。例如在上节显示结果中，net1.outputConnect=[1]和 net2.outputConnect=[0 1]，说明 net1 唯一的一个网络层也为输出层，net2 的第二个网络层为输出层，而第一个网络层不作输出层。

net.outputConnect 属性值一旦改变，网络层输出层的数目（net.numOutputs）和输出层结构细胞矩阵（net.outputs）将随之改变。

7）targetConnect 属性

net.targetConnect 属性定义各网络层是否和目标向量有关，其值为 $1\times N_l$ 的布尔型向量（0 或 1），N_l 为网络层数（net.numLayers）。

可以通过访问 net.targetConnect(i)的值，查看第 i 个网络层是否和目标向量有关。例如在上节显示结果中，net1.targetConnect=[1]和 net1.outputConnect=[0 1]，说明 net1 唯一的一个网络层

和目标向量有关；net2 的第二个网络层和目标向量有关，而第一个网络层和目标向量没有关系。

8）numTargets 属性（只读）

net.numTargets 属性值为目标向量的数目，它等于 targetConnect 矩阵中元素值为 True 的个数之和，即

numTargets=sum(net.targetConnect)

9）numOutputs 属性（只读）

net.numOutputs，该属性值为输出向量的数目，它等于 outputConnect 矩阵中元素值为 1（True）的个数之和，即

numOutputs=sum(net.outputConnect)

10）numInputDelays 属性（只读）

net.numInputDelays 属性定义进行网络仿真时输入向量的延迟量。其值总是设置为与网络输入相连接的权值延迟量的最大值，即

```
numInputDelays=0;
for i=1:net.numLayers
    for j=1 : net.numInputs
        if net.inputConnect(i,j)
            numInputDelays=max(...
                [numInputDelays net.inputWeights{i,j}.delays]);
        end
    end
end
```

例如在上节显示结果中，net1.numTargets 和 net2.numTargets 均为 0，说明输入向量无延迟。

11）numLayerDelays 属性（只读）

net.numLayerDelays 属性定义进行网络仿真时网络层输出单元的延迟量。其值总是设置为与网络层相连接的权值延迟量的最大值，即

```
numLayperDelays=0;
for i=1:net.numLayers
    for j=1:net.numLayers
        if net.layerConnect(i,j)
            numLayerDelays=max(...
                [numLayerDelays net.layerWeights{i,j}.delays]);
        end
    end
end
```

例如在上节显示结果中，net1.numTargets 和 net2.numTargets 均为 0，说明网络层输出单元无延迟。

2. 子对象结构属性

子对象结构属性定义了下列细胞矩阵：每个网络的输入向量结构、网络层结构、输出向量结构、目标向量结构、阈值向量结构和权值向量结构细胞矩阵。

1）inputs 属性（输入层结构）

net.inputs 属性定义网络输入向量的结构，其值为关于网络各输入向量结构的细胞矩阵，大小为 $N_i \times 1$，N_i 为输入向量数（net.numInputs）。

例如在上节显示结果中，net1.inputs 和 net2.inputs 均为{1×1 cell}，说明网络输入向量的结构是 1×1 的细胞矩阵，即只有一个输入向量

net1.inputs=net2.inputs={[1×1 struct]}

每个输入向量的属性结构可以通过 net.inputs{i}进行访问。例如在上节显示结果中

$$\text{net1.inputs\{1\}}=\begin{bmatrix}\text{range:[3×2 double]}\\ \text{size:3}\\ \text{userdata:[1×1 struct]}\end{bmatrix}$$

$$\text{net2.inputs\{1\}}=\begin{bmatrix}\text{range:[2×2 double]}\\ \text{size:2}\\ \text{userdata:[1×1 struct]}\end{bmatrix}$$

每个输入向量的属性值，可以通过

net.inputs{i}.属性名

进行访问。例如

$$\text{net1.inputs\{1\}.range}=\begin{bmatrix}1 & 2\\ -1 & 1\\ 0 & 1\end{bmatrix}$$

2）layers 属性

net.layers 属性定义各网络层的结构特性，其值为关于网络层结构的细胞矩阵，大小为 N_l×1，N_l 为网络层数（net.numLayers）。

例如在上节显示结果中，net1.layers 为{1×1 cell}，说明网络层结构是 1×1 的细胞矩阵，即只有一个网络层；net2.layers 为{2×1 cell}，说明网络层结构是 2×1 的细胞矩阵，即有两个网络层

$$\text{net1.layers} = \{[1\times1\ \text{struct}]\}$$

$$\text{net2.layers} = \begin{bmatrix}[1\times1\ \text{struct}]\\ [1\times1\ \text{struct}]\end{bmatrix}$$

每个网络层的结构属性可以通过 net.layers{i}访问。例如

$$\text{net1.layers\{1\}}=\begin{bmatrix}\text{dimensions:2}\\ \text{distanceFcn:}\\ \text{distances:[]}\\ \text{initFcn:'initwb'}\\ \text{netInputFcn:'netsum'}\\ \text{positions:[0 \quad 1]}\\ \text{size:2}\\ \text{topologyFcn:'hextop'}\\ \text{transferFcn:'hardlim'}\\ \text{userdata:[1×1 struct]}\end{bmatrix}$$

每个网络层的结构属性值，可以通过

net.layers{i}.属性名

访问。例如

net1.layper{1}.transferFcn=hardlim

3）outputs 属性

net.outputs 属性定义各网络层作为输出向量的结构特性，其值为关于输出向量结构的细胞矩阵，大小为 $N_l \times 1$，N_l 为网络层数（net.numLayers）。

每个输出向量的结构属性可以通过 net.outputs{i}访问。例如

$$\text{net1.outputs\{1\}}=\begin{bmatrix} \text{size:2} \\ \text{userdata:[1×1 struct]} \end{bmatrix}$$

每个输出向量的结构属性值，可以通过

net.outputs{i}.属性名

访问。例如

net1.outputs{1}.size=2

4）targets 属性

net.targets 属性定义各网络层目标向量的结构特性，其值为关于目标向量结构的细胞矩阵，大小为 $N_l \times 1$，N_l 为网络层数（net.numLayers）。

每个网络层目标向量的结构属性可以通过 net.targets{i}访问。例如

$$\text{net2.targets\{2\}}=\begin{bmatrix} \text{size:2} \\ \text{userdata:[1×1 struct]} \end{bmatrix}$$

每个目标向量的结构属性值，可以通过

net.targets{i}.属性名

访问。例如

net2.targets{2}.size=2

5）biases 属性

net.biases 属性定义各网络层阈值向量的结构特性，其值为关于阈值向量结构的细胞矩阵，大小为 $N_l \times 1$，N_l 为网络层数（net.numLayers）。

例如在上节显示结果中，net1.biases 为{1×1 cell} containing 1 bias，说明阈值向量结构是 1×1 的细胞矩阵，只有一个阈值向量；net2.layers 为{2×1 cell} containing 2 biases，说明阈值向量结构是 2×1 的细胞矩阵，即有两个网络层，都具有阈值向量

$$net1.biases=\{[1\times1\ struct]\}$$

$$\text{net2.biases}=\begin{Bmatrix} [1\times1\ \text{struct}] \\ [1\times1\ \text{struct}] \end{Bmatrix}$$

每个网络层阈值向量的结构属性可以通过 net.biases{i}访问。例如

$$\text{net2.biases}\{2\} = \begin{bmatrix} \text{initFcn}:'' \\ \text{learn}:1 \\ \text{learnFcn}:\text{'learngdm'} \\ \text{learnParam}:[1\times1\ \text{struct}] \\ \text{size}:2 \\ \text{userdata}:[1\times1\ \text{struct}] \end{bmatrix}$$

每个阈值向量的结构属性值，可以通过

net.biases{i}.属性名

访问。例如

net2.biases{2}.learnFcn=learngdm

6）inputWeights 属性

net.inputWeights 属性定义输入层权值向量的结构特性，其值为关于输入层权值向量结构的细胞矩阵，大小为 $N_l \times N_i$，N_l 为网络层数（net.numLayers），N_i 为输入向量数（net.numInputs）。

例如在上节显示结果中，net1.inputWeights 为{1×1 cell} containing 1 input weight，说明输入层的权值向量结构是 1×1 的细胞矩阵，只有一个输入层权值向量；net2.layers 为{2×1 cell} containing 1 input weights，说明阈值向量结构是 2×1 的细胞矩阵，即有两个网络层，但只有一个输入层权值向量

$$\text{net1.inputWeights} = \{[1\times1\ \text{struct}]\}$$
$$\text{net2.inputWeights} = \begin{bmatrix} [1\times1\ \text{struct}] \\ \\ \end{bmatrix}$$

可以通过访问 net.inputWeights{i,j}，了解从第 j 个输入向量连接到第 i 个网络层的结构属性。例如

$$\text{net2.inputWeights}\{1,1\} = \begin{bmatrix} \text{delays}:0 \\ \text{initFcn}:'' \\ \text{learn}:1 \\ \text{learnFcn}:\text{'learngdm'} \\ \text{learnParam}:[1\times1\ \text{struct}] \\ \text{size}:[15\quad 2] \\ \text{userdata}:[1\times1\ \text{struct}] \\ \text{weightFcn}:\text{'dotprod'} \end{bmatrix}$$

每个输入层权值向量的结构属性值，可以通过

net.inputWeights{i,j}.属性名

访问，例如

net2.inputWeights{1,1}.learnFcn=learngdm

7）layerWeights 属性

net.layerWeights 属性定义各网络层权值向量的结构特性，其值为关于各网络层权值向量结构的细胞矩阵，大小为 $N_l \times N_l$，N_l 为网络层数（net.numLayers）。

例如在上节显示结果中，net1.layerWeights 为{1×1 cell} containing no layer weights，说明网络层的权值向量结构是 1×1 的细胞矩阵，但没有网络层权值向量；net2.layerWeights 为{2×2 cell} containing 1 layer weight，说明网络层权值向量结构是 2×2 的细胞矩阵，即有两个网络层，但只有一个网络层权值向量

$$\text{net1.layerWeights} = \{[\,]\}$$

$$\text{net2.layerWeights} = \begin{bmatrix} [\,] & [\,] \\ [1\times1\ \text{struct}] & [\,] \end{bmatrix}$$

net2.layerWeight{2,1}不为空值，说明有第 1 网络层到第 2 网络层的权值向量。网络层权值向量的结构属性可以由 net.layerWeights{i,j}查看。例如

$$\text{net2.layerWeights}\{2,1\} = \begin{bmatrix} \text{delays}:0 \\ \text{initFcn}:'' \\ \text{learn}:1 \\ \text{learnFcn}:\text{'learngdm'} \\ \text{learnParam}:[1\times1\ \text{struct}] \\ \text{size}:[2\quad 15] \\ \text{userdata}:[1\times1\ \text{struct}] \\ \text{weightFcn}:\text{'dotprod'} \end{bmatrix}$$

每个网络层权值向量的结构属性值，可以通过

net.layerWeights{i,j}.属性名

访问。例如

net2.layerWeights{2,1}learnFcn=learngdm

3. 函数属性

函数属性定义了一个网络在进行权值/阈值调整、初始化、误差性能计算或训练时采用的算法。

1）initFcn 属性

net.initFcn 属性定义了网络初始化权值/阈值向量时所采用的函数，它可以被设置为任意一个进行网络权值/阈值初始化的函数名，包括 initlay（网络层初始化函数）工具箱函数。

init 函数一旦被调用，就可以实现网络权值/阈值的初始化：

net=init(net)

另外，用户可以自定义权值/阈值初始化函数。

init 属性值一旦发生变化，网络的初始化参数（net.initParam）将被设置为新的初始化函数所包含的参数及其默认参数值。

2）adaptFcn 属性

net.adaptFcn 属性定义了网络进行权值/阈值调整所采用的函数，它可以被设置为任意一个进行权值/阈值调整的函数名，包括 trains 函数。

adapt 函数一旦被调用，就可以实现网络权值/阈值的调整

[net, Y, E, Pf, Af]=adapt(NET, P, T, Pi, Ai)

另外，用户可以自定义权值/阈值调整函数。

adaptFcn 属性值一旦发生变化，网络的调整参数（net.adaptParam）将被设置为新的调整函数所包含的参数及其默认参数值。

3）performFcn 属性

net.performFcn 属性定义了网络用于衡量网络性能所采用的函数，它可以被设置为任意一个网络性能函数名，MATLAB 工具箱误差性能函数如表 2-1 所示。

表 2-1　MATLAB 工具箱误差性能函数

函 数 名	说　　明
mae	绝对平均误差性能函数(mean absolute error)
mse	均方误差性能函数(mean squared error)
msereg	归一化均方误差性能函数(mean squared error with regularization)
sse	平方和误差性能函数(sum squared error)

当调用 train 函数时，上述性能函数被用于训练过程中的性能计算

[net, tr]=train(NET, P, T, Pi, Ai)

另外，用户可以自定义性能函数。

performFcn 属性值一旦发生变化，网络性能参数（net.performParam）将被设置为新的性能函数所包含的参数及其默认参数值。

4）trainFcn 属性

net.trainFcn 属性定义了网络用于训练网络性能所采用的函数，它可以被设置为任意一个训练函数名，MATLAB 工具箱的训练函数如表 2-2 所示。

表 2-2　MATLAB 工具箱的训练函数

函　数　名	说　　明
trainbfg	BFGS 算法(拟牛顿反向传播算法)训练函数
trainbr	贝叶斯归一化法训练函数
traincgb	Powell-Beale 共轭梯度反向传播算法训练函数
traincgf	Fletcher-Powell 变梯度反向传播算法训练函数
traincgp	Polak-Ribiere 变梯度反向传播算法训练函数
traingd	梯度下降反向传播算法训练函数
traingda	自适应调整学习率的梯度下降反向传播算法训练函数
traingdm	附加动量因子的梯度下降反向传播算法训练函数
traingdx	自适应调整学习率并附加动量因子的梯度下降反向传播算法训练函数
trainlm	Levenberg-Marquardt 反向传播算法训练函数

续表

函　数　名	说　　明
trainoss	OSS(one-step secant)反向传播算法训练函数
trainrp	RPROP(弹性 BP 算法)反向传播算法训练函数
trainscg	SCG(scaled conjugate gradient)反向传播算法训练函数
trianb	以权值/阈值的学习规则采用批处理的方式进行训练的函数
trainc	以学习函数依次对输入样本进行训练的函数
trainr	以学习函数随机对输入样本进行训练的函数

当调用 train 函数时，上述训练函数被用于训练网络

[net, tr]=train(NET, P, T, Pi, Ai)

另外，用户可以自定义训练函数。

trainFcn 属性值一旦发生变化，网络训练参数（net.trainParam）将被设置为新的训练函数所包含的参数及其默认参数值。

4．权值和阈值

权值和阈值属性定义了网络的可调整参数：权值向量和阈值向量。

1）IW 属性

net.IW 属性定义从网络输入向量到网络层的权值向量（即输入层的权值向量）结构。其值为 $N_l \times N_i$ 的细胞矩阵，N_l 为网络层数（net.numLayers），N_i 为输入向量数（net.numInputs）。例如上节显示结果中，net1.IW 为{1×1 cell} containing 1 input weight matrix，说明输入层的权值向量结构是 1×1 的细胞矩阵，有一个输入层权值向量；net2.IW 为{2×1 cell} containing 1 input weight matrix，说明输入层的权值向量结构是 2×1 的细胞矩阵，有一个输入层权值向量。net.IW 的元素值说明了各输入层权值向量的结构和数值类型

$$\text{net1.IW} = \{[2\times 3\ \text{doulbe}]\}$$

$$\text{net2.IW} = \begin{Bmatrix} [15\times 2\ \text{double}] \\ [] \end{Bmatrix}$$

通过访问 net.IW{i,j}，可以获得第 i 个网络层来自第 j 个输入向量的权值向量值。例如

$$\text{net1.IW}\{1,1\} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

2）b 属性

net.b 属性定义各网络层的阈值向量结构。其值为 $N_l \times 1$ 的细胞矩阵，N_l 为网络层数（net.numLayers）。例如在上一节显示结果中，net1.b 为{1×1 cell} containing 1 bias vector，说明阈值向量结构是 1×1 的细胞矩阵，有一个阈值向量；net2.b 为{2×1 cell} containing 2 bias vectors，说明阈值向量结构是 2×2 的细胞矩阵，含有两个阈值向量。net.b 的元素值说明了各网络层阈值向量的结构和数值类型

$$net1.b=\{[2\times 1\ \text{double}]\}$$

$$net2.b=\begin{Bmatrix}[15\times 1\ \text{double}]\\ [2\times 1\ \text{double}]\end{Bmatrix}$$

通过访问 net.b{i}，可以获得第 i 个网络层阈值向量。

3）LW 属性

net.LW 属性定义从一个网络层到另一个网络层的权值向量结构。其值为 $N_l\times N_l$ 的细胞矩阵，N_l 为网络层数。例如在上节显示结果中，net1.LW 为{1×1 cell} containing no layer weight matrices，说明网络层的权值向量结构是 1×1 的细胞矩阵，但没有网络层之间的权值向量；net2.IW 为{2×2 cell} containing 1 layer weight matrix，说明输入层的权值向量结构是 2×2 的细胞矩阵，含有一个网络层之间的连接权值向量。net1.LW 的元素值说明了各网络层权值向量的结构和数值类型

$$net1.LW=\{[\,]\}$$

$$net2.LW=\begin{Bmatrix}[\,] & [\,]\\ [2\times 15\ \text{double}] & [\,]\end{Bmatrix}.$$

只有 net.LW{2,1}不为空值，说明只有第 2 网络层来自第 1 网络层的权值向量。

通过访问 net.LW{i,j}，可以获得第 i 个网络层来自第 j 个网络层的权值向量值。

5．参数属性

1）adaptParam 属性

net.adaptParam 属性定义当前网络权值/阈值调整函数的参数及参数值，它取决于当前的权值/阈值调整函数（net.adaptFcn），可以查看有关调整函数的帮助获得这些参数及参数值。在 MATLAB 的命令窗口输入命令

help(net.adaptFcn)

也可以获得这些参数及参数值的具体描述。

2）initParam 属性

net.initParam 属性定义当前初始化函数参数及参数值，它取决于当前的初始化函数（net.initFcn），可以查看有关初始化函数的帮助获得这些参数及参数值。在 MATLAB 命令窗口输入命令

help(net.initFcn)

也可以获得这些参数及参数值的具体描述。

3）performParam 属性

net.performParam 属性定义当前性能函数的参数及参数值，它取决于当前的性能函数（net.performFcn），可以查看有关性能函数的帮助获得这些参数及参数值。在 MATLAB 命令窗口中输入命令

help(net.performFcn)

也可以获得这些参数及参数值的具体描述。

4）trainParam 属性

net.trainParam 属性定义当前训练函数的参数及参数值，取决于当前的训练函数（net.trainFcn），

可以查看有关训练函数的帮助获得这些参数及参数值。在 MATLAB 命令窗口输入命令

help(net.trainFcn)

也可以获得这些参数及参数值的具体描述。

6．其他属性

userdata 属性

net.userdata 属性为用户提供了增加关于网络对象的用户信息的地方，它预先只定义了一个字段，其值为一个提示信息

net.userdata=note:'Put your custom network information here'

用户可以通过修改 net.userdata.note 的值，增加关于网络对象的用户信息。

2.5.2　子对象属性

1．输入向量

net.inputs，该子对象的属性详细定义了网络的每个输入向量的每一个输入量。例如

net1.inputs=net2.inputs={[1×1 struct]}

通过访问 net.inputs{i}可以获得第 i 个输入向量的属性值

$$\text{net1.inputs}\{1\}=\begin{bmatrix}\text{range}:[3\times 2\ \text{double}]\\ \text{size}:3\\ \text{userdata}:[1\times 1\ \text{struct}]\end{bmatrix}$$

$$\text{net2.inputs}\{1\}=\begin{bmatrix}\text{range}:[2\times 2\ \text{double}]\\ \text{size}:2\\ \text{userdata}:[1\times 1\ \text{struct}]\end{bmatrix}$$

1）range 属性

net.inputs{i}.range 定义了第 i 个输入向量中每个元素的取值范围，其值是一个 R×2 的矩阵，R 为输入向量的元素个数。矩阵的第一列为每个元素的最小值，第二列为每个元素的最大值。它包含了两个信息：①输入向量 R 的元素个数（亦即输入变量的个数）；②每个输入变量的取值区间。这些信息确定了输入向量的规模，另外 range 属性值还用于在一些初始化函数中确定连接输入向量的权值和阈值的初值。

当 net.inputs{i}.range 的行数变化时，表明输入向量的元素数目发生了改变，那么网络输入向量的 size 属性值（net.inputs{i}.size）、与之相连接的权值的 size 属性值（net.inputWeights{: ,i}.size）及输入权值向量（net.IW{: , i}）的大小的会自动作相应的变化。

2）size 属性

Net.inputs{i}.size 定义了网络各输入向量的元素数目，可被设置为 0 或正整数。当其值发生变化时，表明输入向量的元素数目发生了变化，那么相应的 range 属性值（net.inputs{i}.range）、与之相连接的权值的 size 属性值（net.inputWeights(:, i).size）及输入权值向量（net.IW{:, i}.size）的大小均会自动作相应的变化。

3）userdata 属性

net.inputs{i}.userdata 和 net.userdata 为用户提供了增加关于输入向量的用户信息的地方，它预先只定义了一个字段，其值为一个提示信息。

net.inputs{i}.userdata=note:'Put your custom input information here.'
用户可以通过修改 net.inputs{i}.userdata.note 的值，增加关于输入向量的用户信息。

2. 网络层

net.layers，该子对象的属性详细定义了网络的每一个网络层。例如

$$\text{net1.layers} = \{[1\times1\ \text{struct}]\}$$

$$\text{net2.layers} = \begin{Bmatrix}[1\times1\ \text{struct}]\\ [1\times1\ \text{struct}]\end{Bmatrix}$$

通过访问 net.layers{i}可以获得第 i 个网络层的属性值

$$\text{net1.layers}\{1\} = \begin{bmatrix} \text{dimensions}: 2 \\ \text{distanceFcn}: '' \\ \text{distances}: [\,] \\ \text{initFcn}: \text{'initwb'} \\ \text{netInputFcn}: \text{'netsum'} \\ \text{positions}: [0\quad 1] \\ \text{size}: 2 \\ \text{topologyFcn}: \text{'hextop'} \\ \text{transferFcn}: \text{'hardlim'} \\ \text{userdata}: [1\times1\ \text{struct}] \end{bmatrix}$$

$$\text{net2.layers}\{2\} = \begin{bmatrix} \text{dimensions}: 2 \\ \text{distanceFcn}: '' \\ \text{distances}: [\,] \\ \text{initFcn}: \text{'initwb'} \\ \text{netInputFcn}: \text{'netsum'} \\ \text{positions}: [0\quad 1] \\ \text{size}: 2 \\ \text{topologyFcn}: \text{'hextop'} \\ \text{transferFcn}: \text{'purelin'} \\ \text{userdata}: [1\times1\ \text{struct}] \end{bmatrix}$$

1）dimensions 属性

net.layers{i}.dimensions 属性定义了第 i 个网络层神经元的维数。对于自组织映射的多维方式，能够设置网络层的神经元维数是很重要的。net.layers{i}.dimensions 可以被设置成所有元素的值为 0 或正整数的行向量，此时，行向量所有元素的乘积即为该网络层的神经元数。

当采用网络层拓扑函数（net.layers{i}.topologyFcn）计算神经元在网络层中的位置（net.layers{i}.positions）时，将用到网络层神经元维数。

net.layers{i}.dimensions 属性一旦改变，网络层的大小（net.layers{i}.size）、网络层神经元的位置（net.layes{i}.positions）以及两个神经元之间的距离（net.layers{i}.distances）都会随之改变。

2）distanceFcn 属性

net.layers{i}.distanceFcn，该属性定义一函数，用于第 i 个网络层中的神经元之间距离的计算，它是根据神经元的位置（net.layers{i}.positions）来进行计算的。神经元的距离用于自组织映射神经网络。

该属性可被设置成神经网络工具箱中的任意一个距离函数名（距离计算函数见表 2-3）。

表 2-3　MATLAB 神经网络工具箱的距离计算函数

函　数	说　明
boxdist	计算两个位置向量之间距离的距离函数
dist	欧几里得(Euclidean)距离权值函数
linkdist	连接距离函数
mandist	曼哈顿(Manhattan)距离权值函数

除了以上距离函数外，用户还可以自定义距离函数，相关内容读者可以参考相关文献。

net.layers{i}.distanceFcn 属性一旦发生改变，网络层神经元之间的距离（net.layers{i}.distances）也将随之改变。

3）initFcn 属性

net.layers{i}.initFcn，如果网络初始化函数（net.initFcn）设置为 initlay，则该属性定义第 i 个网络层的初始化函数。

该属性可被设置为任意一个神经网络工具箱中的网络层初始化函数名（网络层初始化函数见表 2-4）。

表 2-4　MATLAB 神经网络工具箱的网络层初始化函数

函　数	说　明
initnw	NW(Nguyen-Window)网络层初始化函数
initwb	通过权值和阈值进行初始化的网络层初始化函数

如果网络初始化函数设置为 initlay，那么，当 init 函数被调用时，该属性定义的网络层初始化函数将用于对网络层的权值和阈值的初始化

net=init(net)

除了以上网络层初始化函数之外，用户还可以自定义网络层初始化函数。

4）distances 属性（只读）

net.layers{i}.distances，该属性定义第 i 个网络层中神经元之间的距离，这些距离用于自组织映射神经网络。其值为网络层距离函数（net.layers{i}.distanceFcn）的计算结果，它是通过网络层神经元的位置（net.layers{i}.positions）进行计算的。

5）positions 属性（只读）

net.layers{i}.positions，该属性定义第 i 个网络层中神经元的位置，这些位置用于自组织映射。其值为网络层拓扑结构函数（net.layers{i}.topologyFcn）关于位置的计算结果，它是通过网络层神经元维数（net.layers{i}.dimensions）进行计算的。

可以用 plostom 函数画出网络层神经元的位置图。例如

plotsom(net2.layers{1}.positions)

net2 第 1 网络层的位置如图 2-3 所示。net2.layers {1}.dimensions=[15]是一维的，所以所有神经元的位置都在水平轴上，即 positions(2,i)=0。

其总的程序代码如下：

```
p=[1 2;-1 1;0 1];
net1=newp(p,2);
net2=newff([-1    1;-1    1],[15    2],{'tansig'
'purelin'},'traingdx','learngdm');
net2.layers{1}.dimensions=[15];
for i=1:15
positions(2,i)=0;
plotsom(net2.layers{1}.positions)
end
```

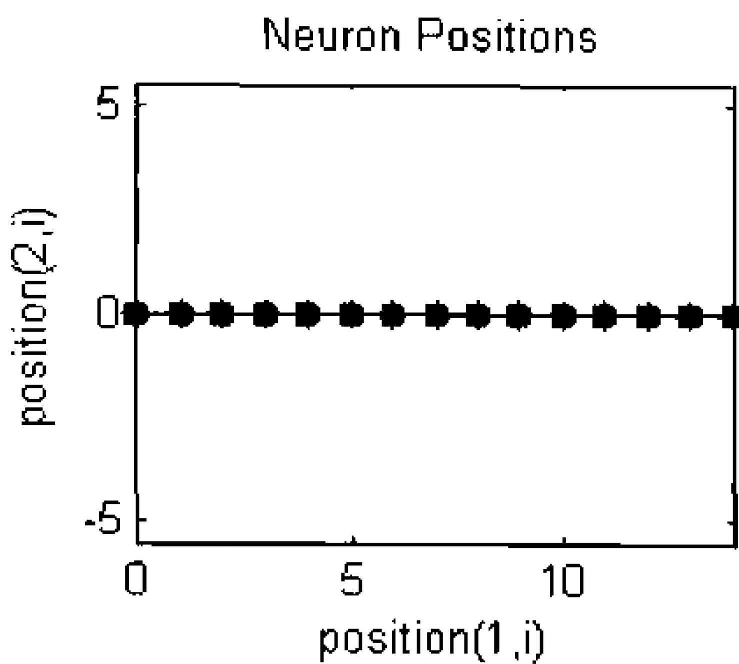


图 2-3　net2 第 1 网络层神经元的位置图

当然，网络层的神经元拓扑结构还可以设置成 n 维的，当 n>3 时，神经元的位置图将由 n 维超立体空间的点表示，此时 plotsom 函数只标出开始的三维位置坐标，图 2-4 是 dimensions=[3 2]的神经元位置示意图，它是二维的；图 2-5 是 dimensions=[1 3 2 1]的神经元位置示意图，它是四维的，图中只标出前三维位置坐标。

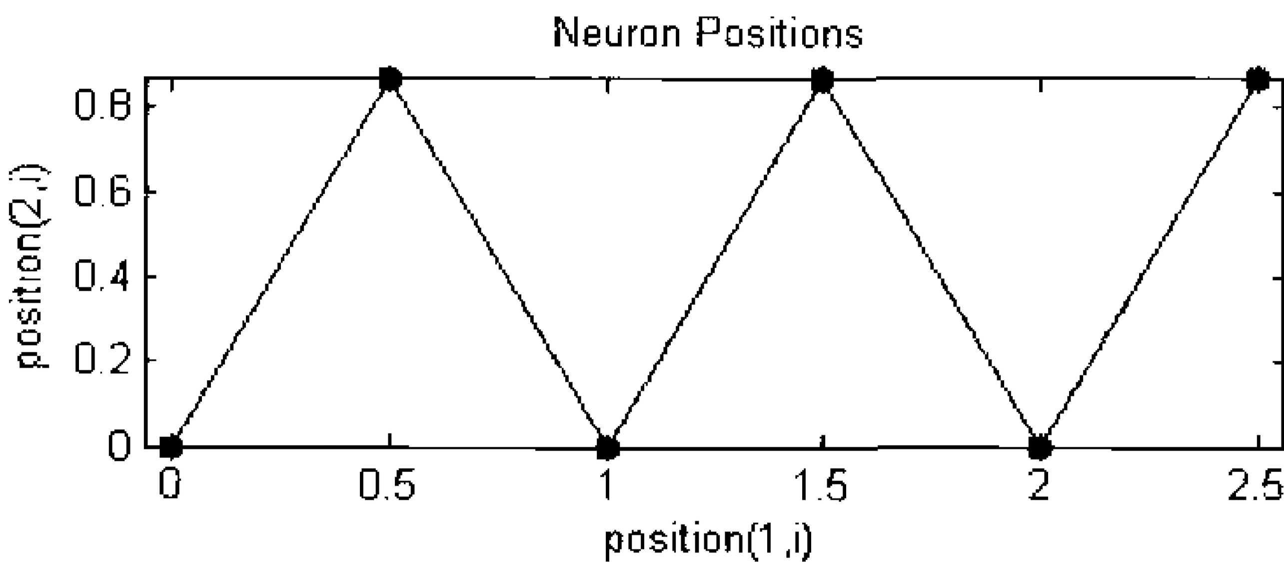


图 2-4　dimensions=[3 2]的神经元位置图

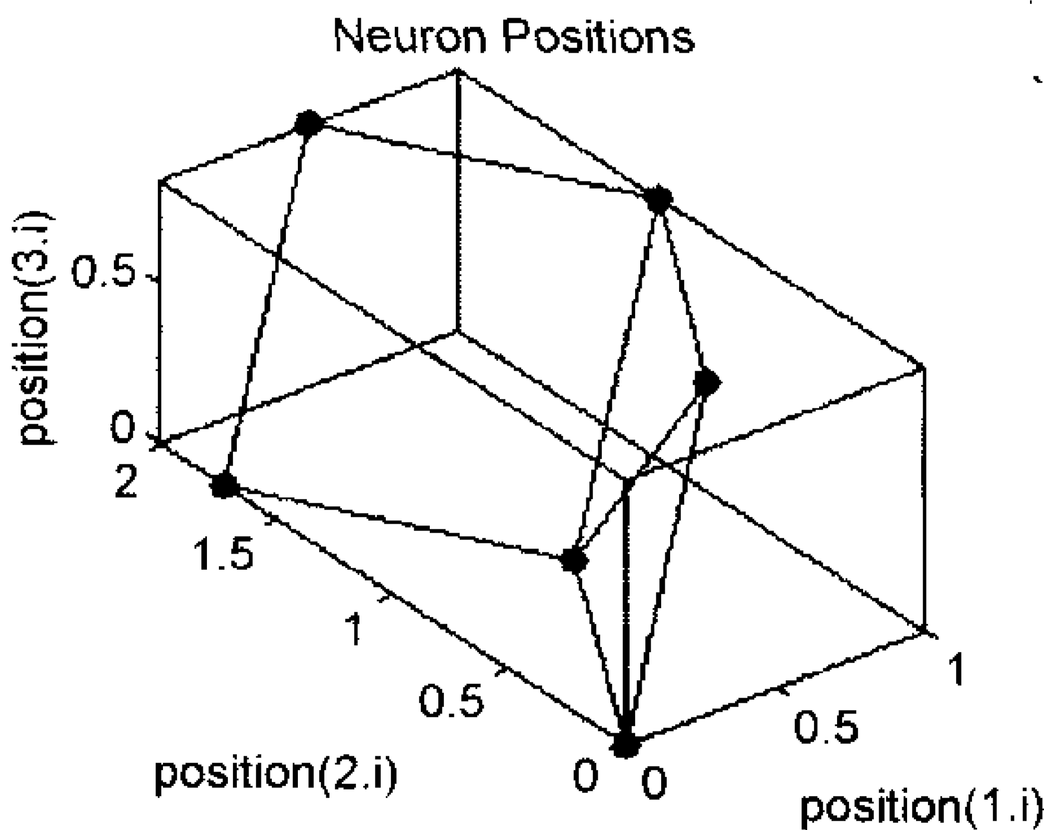


图 2-5　dimensions=[1 3 2 1]的神经元位置图

6）netInputFcn 属性

net.layers{i}.netInputFcn 属性定义一个网络输入函数，以给定的权值和阈值计算第 i 个网络层的输入。

该属性可被设置为任意一个神经网络工具箱中的网络层输入函数名（网络输入函数见表 2-5）。

表 2-5　神经网络工具箱的网络输入函数

函　数	说　明
netprod	求积网络输入函数
netsum	求和网络输入函数

当 sim 函数被调用时，输入函数被用于网络的仿真

[Y, Pf, Af]=sim(net, P, Pi, Ai)

除了以上网络输入函数之外，用户还可以自定义网络输入函数。

7）topologyFcn 属性

net.layers{i}.topologyFcn，该属性定义一拓扑结构函数，用以计算第 *i* 个网络层中的神经元位置（net.layers{i}.positions），该位置是通过网络层神经元的维数 net.layers{i}. dimensions 进行计算的。其值可以设置成神经网络工具箱中任意一拓扑结构函数名（拓扑结构函数见表 2-6）。

表 2-6 神经网络工具箱的拓扑结构函数

函　数	说　明
gridtop	网状网络层拓扑结构函数
hextop	六边形网络层拓扑结构函数
randtop	随机网络层拓扑结构函数

除了以上拓扑结构函数外，用户还可以自定义拓扑结构函数，相关内容读者可以自行参考相关文献。

topologyFcn 属性值一旦改变，则网络层神经元的位置（net.layers{i}.positions）也会随之更新。

8）size 属性

net.layers{i}.size，该属性定义第 *i* 个网络层中的神经元数目，其值可以设置为 0 或正整数。

其值一旦发生变化，则所有的 net.inputWeights{i, :}.size（连接到该网络层的输入权值向量元素的数目）、net.layerWeights{:, i}.size（该网络层连接到其他网络层权值向量元素的数目）、net.layerWeights{i, :}.size（其他网络层连接到该网络层权值向量元素的数目）、net.biases{i}.size（该网络层阈值向量元素的数目）等都将随之改变。

与之相关的 net.IW{i, :}、net.LW{i, :}、net.LW{:, i}和 biases(net.b{i})向量的维数也会随之改变。

如果网络层具有输出向量和目标向量，则 net.outputs{i}.size（网络层输出向量元素的数目）和 net.targets{i}.size（目标向量元素的数目）也会改变。

同时，网络层神经元的维数（net.layers{i}.dimensions）将被设置成与该属性相同的值，这仅适用于神经元的维数为一维的情况；对于多维的情况，应该直接设置 net.layers{i}. dimensions，而不应使用 size 属性设置。

9）userdata 属性

net.layers{i}.userdata，该属性值为用户提供增加关于网络层向量的用户信息的地方，它预先只定义一个字段，其值为一个提示信息

net.layers{i}.userdata=note:'Put your custom layer information here.'

用户可以通过修改 net.layers{i}.userdata.note 的值，增加关于网络层向量的用户信息。

10）transferFcn 属性

net.layers{i}.transferFcn，该属性定义网络的传输函数，用以计算第 *i* 个网络层的输出，该输出是通过给定的网络层输入值进行计算的。其值可以设置成神经网络工具箱中任意一个传输函数名（传输函数见表 2-7）。

表 2-7 神经网络工具箱的传输函数

函　数	说　明
compet	竞争型传输函数
hardlim	阈值型传输函数

续表

函　数	说　明
hardlims	对称阈值型传输函数
logsig	S 形传输函数
poslin	正线性传输函数
purelin	线性传输函数
radbas	径向基传输函数
satlin	饱和线性传输函数
satlins	对称饱和线性传输函数
softmax	柔性最大值传输函数
tanhsig	双曲正切 S 形传输函数
tribas	三角形径向基传输函数

除了表中的传输函数外，用户还可以自定义传输函数，相关内容在后面介绍。当 sim 函数被调用时，传输函数被用于网络仿真

[Y, Pf, Af]=sim(net, P, Pi, Ai)

3．输出向量

net.outputs，该子对象的属性详细定义了网络的每一个输出向量，例如

$$\text{net1.outputs}=\{[1\times 1\ \text{struct}]\}$$

$$\text{net2.outputs}=\{[]\quad [1\times 1\,\text{struct}]\}$$

每个输出向量的结构属性值，可以通过

net.outputs{i, j}

访问。例如

$$\text{net2.outputs}\{1,2\}=\begin{bmatrix}\text{size}:2\\ \text{userdata}:[1\times 1\,\text{struct}]\end{bmatrix}$$

1）size 属性（只读）

net.outputs{i}.size，该属性定义了第 i 个网络层输出向量中元素的数目，其值为第 i 个网络层神经元的数目（net.layers{i}.size）。

2）userdata 属性

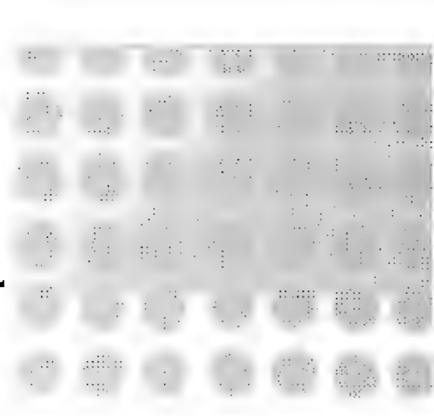
net.outputs{i}.userdata，该属性为用户提供了增加关于第 i 个网络层输出向量的用户信息的地方，它预先只定义一个字段，其值为一个提示信息

net.outputs{i}.userdata=note:'Put your custom output information here.'

用户可以通过修改 net.outputs{i}.userdata.note 的值，增加关于第 i 个网络层输出向量的用户信息。

4．阈值向量

net.biases，该子对象的属性详细定义网络的每一个阈值向量。例如



$$net1.biases = \{[1\times 1\ struct]\}$$

$$net2.biases = \begin{bmatrix} [1\times 1\ struct] \\ [1\times 1\ struct] \end{bmatrix}$$

每个阈值向量的结构属性值，可以通过

net.biases{i}

访问。例如

$$net2.biases\{1\} = \begin{bmatrix} initFcn: '' \\ learn: 1 \\ learnFcn: 'learngdm' \\ learnParam: [1\times 1\ struct] \\ size: 15 \\ userdata: [[1\times 1\ struct] \end{bmatrix}$$

1）initFcn 属性

net.biases{i}.initFcn，该属性定义第 i 个网络层阈值向量的初始化函数，如果网络的初始化函数为 initlay，则第 i 个网络层阈值向量的初始化函数的函数值为 initwb。其值可以设置成神经网络工具箱中任意一个阈值初始化函数名（初始化函数见表 2-8）。

表 2-8　神经网络工具箱的初始化函数

函　　数	说　　明
initcon	"良心"(conscience)阈值初始化函数
initzero	零权值/阈值初始化函数
rands	对称随机权值/阈值初始化函数

除了表中的初始化函数外，用户还可以自定义初始化函数，当 init 函数被调用时，initFcn 定义的函数将对第 i 个网络层阈值向量（net.b{i}）进行初始化

net=init(net)

2）learn 属性

net.biases{i}.learn，该属性定义第 i 个阈值向量在训练和调整过程中是否变化。其值可以设置为 0 或 1。它容许或禁止在训练（train）和调整（adapt）过程中对阈值向量进行学习

[net,Y, E, Pf, aF]=adapt(NET, P, T, Pi, Ai)

[net,tr]=train(NET, P, T, Pi, Ai)

3）learnParam 属性

net.biases{i}.learnParam，该属性定义了第 i 个网络层阈值向量当前的学习函数的参数及参数值，其值取决于当前的学习函数（net.biases{i}.learnFcn），可以查阅当前学习函数的参考文献帮助获得当前学习函数的具体描述

help(net.biases{i}.learnFcn)

4）learnFcn 属性

net.baises{i}.learnFcn，如果网络的训练函数是 trainb、trainc 和 trainr，或者网络调整函数为 trains，则该属性定义第 i 个网络层阈值向量在训练和调整过程中的学习函数。其值可以设置成神经网络工具箱中任意一个学习函数名（学习函数见表 2-9）。

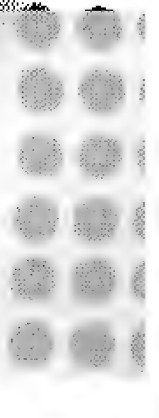


表 2-9　神经网络工具箱的学习函数

函　　数	说　　明
learncon	“良心”(conscience)阈值学习函数
learngd	梯度下降权值/阈值学习函数
learngdm	附加动量因子的梯度下降权值/阈值学习函数
learnp	感知器权值/阈值学习函数
learnpn	归一化感知器权值/阈值学习函数
learnwh	WH(Widrow-Hoff)权值/阈值学习规则

除了以上学习函数外，用户还可以自定义学习函数。如果网络的训练函数是 trainb、trainc 和 trainr，或者网络调整函数为 trains，则当 train 函数被调用时，learnFcn 定义的函数将对第 i 个网络层阈值向量（net.b{i}）进行更新

[net, Y, E, Pf, Af]=adapt(NET, P, T, Pi, Ai)

[net,tr]=train(NET, P, T, Pi, Ai)

该属性值一旦改变，则阈值向量的学习参数（net.biases{i}.learnFcn）将被新定义的学习函数的参数及其默认参数值代替。

5）size 属性（只读）

net.biases{i}.size，该属性定义了第 i 个网络层阈值向量元素的数目，其值为第 i 个网络层神经元的数目（net.layers{i}.size）。

6）userdata 属性

net.baises{i}.userdata，该属性为用户提供了增加关于第 i 个网络层阈值向量的用户信息，可以通过修改 net.biases{i}.userdata.note 的值，增加关于第 i 个阈值向量的用户信息。

5．输入权值向量

net.inputWeights，该子对象的属性详细定义网络的每一个输入权值向量。例如

$$\text{net1.inputWeights} = \{[1\times1\ \text{struct}]\}$$

$$\text{net2.inputWeights} = \begin{Bmatrix} [1\times1\ \text{struct}] \\ [\,] \end{Bmatrix}$$

每个输入权值向量的结构属性值，可以通过

net.inputWeights{i,j}

访问。例如

$$\text{net1.inputWeights}\{1\}=\begin{bmatrix} \text{delays}:0 \\ \text{initFcn}:\text{'initzero'} \\ \text{learn}:1 \\ \text{learnFcn}:\text{'learnp'} \\ \text{learnParam}:[\,] \\ \text{size}:[2\quad 3] \\ \text{userdata}:[1\times1\ \text{struct}] \\ \text{weightFcn}:\text{'dotprod'} \end{bmatrix}$$

1）delays 属性

net.inputWeights{i,j}.delays，该属性定义第 j 个输入向量与第 i 个网络层权值之间的抽头延迟线，其值为从 0 或正整数开始的逐渐增大的行向量。

该属性值一旦改变，权值向量的大小（net.inputWeights{i,j}.size）和权值向量维数（net.IW{i,j}）将被更新。

2）initFcn 属性

net.inputWeights{i,j}.initFcn，如果网络的初始化函数为 initlay，且网络层的初始化函数为 initwb，则该属性定义第 j 个输入向量与第 i 个网络层权值向量的初始化函数。其值可以设置成神经网络工具箱中任意一个权值初始化函数名（权值初始化函数如表 2-10 所示）。

表 2-10　神经网络工具箱的权值初始化函数

函　数	说　明
initzero	零权值/阈值初始化函数
midpoint	中点权值初始化函数
randnc	归一化列权值初始化函数
randnr	归一化行权值初始化函数
rands	对称随机权值/阈值初始化函数

除了以上权值初始化函数外，用户还可以自定义权值初始化函数。如果网络的初始化函数（net.initFcn）为 initlay，且网络层的初始化函数（net.layers{i}.initFcn）为 initwb，则当 init 函数被调用时，net.inputWeights{i,j}.initFcn 所定义的函数将用于计算从第 j 个输入向量到第 i 个网络层的初始权值向量值

net=init(net)

3）learn 属性

net.inputWeights{i,j}.learn，该属性定义第 j 个输入向量与第 i 个网络层的权值向量在训练和调整过程中是否改变，其值可设置为 0 或 1，分别表示权值向量在训练和调整过程中禁止和容许进行学习

[net, Y, E, Pf, Af]=adapt(NET, P, T, Pi, Ai)

[net,tr]=train(NET, P, T, Pi, Ai)

4）learnFcn 属性

net.inputWeights{i,j}.learnFcn，如果网络的训练函数为 trainb、trainc 或 trainr，或网络的调整函数为 trains，则该属性定义第 j 个输入向量与第 i 个网络层权值向量的学习函数。其值可以设置成神经网络工具箱中任意一个权值学习函数名（权值学习函数见表 2-11）。

表 2-11　神经网络工具箱的权值学习函数

函　数	说　明
learngd	梯度下降权值/阈值学习函数
learngdm	附加动量因子的梯度下降权值/阈值学习函数
learnh	Hebb 权值学习函数
learnhd	带衰减因子的 Hebb 权值学习函数

续表

函　数	说　明
learnis	内星权值学习函数
learnk	Kohonen 权值学习函数
learnlv1	LVQ1 权值学习函数
learnlv2	LVQ2 权值学习函数
learnos	外星权值学习函数
learnp	感知器权值/阈值学习函数
learnpn	归一化感知器权值/阈值学习函数
learnsom	自组织映射权值学习函数
learnwh	WH(Widrow-Hoff)权值/阈值学习规则

除了以上权值初始化函数外，用户还可以自定义权值初始化函数。如果网络的训练函数（net.trainFcn）为 trainb、trainc 或 trainr，或网络的调整函数（net.adaptFcn）为 trains，则 i 当 train 函数被调用时，net.inputWeights{i,j}.learn 所定义的函数将用于计算第 j 个输入向量与第 i 个网络层的权值向量

[net, Y, E, Pf, Af]=adapt(NET, P, T, Pi, Ai)

[net,tr]=train(NET, P, T, Pi, Ai)

5）learnParam 属性

net.inputWeights{i,j}.learnParam，该属性定义第 i 个网络层来自第 j 个输入向量的权值向量学习函数的参数及参数值，其值取决于当前的学习函数（net.inputWeights{i,j}. learnFcn），可以查阅当前学习函数的帮助以获得当前学习函数的具体描述

help(net.inputWeights{i,j}.learnFcn)

6）size 属性（只读）

net.inputWeights{i,j}.size，该属性定义第 i 个网络层与第 j 个输入向量的连续权的数目，其值为具有两个元素的行向量，分别表示相应权值向量（net.IW{i,j}）的行数与列数。第一个元素的值为第 i 个网络层的神经元的数目（net.layers{i}.size），第二个元素的值为第 j 个输入向量元素的数目（net.inputs{j}.size）与第 i 个网络层输入抽头延迟线长度的乘积

length(net.inputWeights{i,j}.delays)*net.inputs{j}.size

7）userdata 属性

net.inputWeights{i,j}.userdata，该属性为用户提供了增加关于第 i 个网络层与第 j 个输入个向量连接权数目的用户信息的地方，它预先只定义一个字段，其值为一个提示信息

net.inputWeights{i,j}.userdata=note:'Put your custom weight information here.'

用户可以通过修改 net.inputWeights{i,j}.userdata.note 的值，增加关于第 i 个网络层与第 j 个输入向量连接权数目的用户信息。

8）weightFcn 属性

net.inputWeights{i,j}.weightFcn，该属性定义第 i 个网络层来自第 j 个输入向量的权值函数。其值可以是神经网络工具箱的任意一个权值函数名（权值函数如表 2-12 所示）。

表 2-12　神经网络工具箱的权值函数

函　　数	说　　明
dist	欧几里得(Euclidean)距离权值函数
dotprod	点积权值函数
mandist	曼哈顿(Manhattan)距离权值函数
negdist	归一化列权值初始化函数
normprod	归一化行权值初始化函数

当 sim 函数被调用时，权值函数被用于网络的仿真

[Y, Pf, Af]=sim(net, P, Pi, Ai)

6. 目标向量

net.targets，该子对象的属性详细定义了网络的每一个目标向量。例如

net.targets={[1×1 struct]}

$$\text{net2.targets} = [[]\quad [1\times 1\ \text{struct}]]$$

每个目标向量的结构属性值，可以通过

net.targets{i,j}

访问。例如

$$\text{net2.targets}\{1.2\} = \begin{Bmatrix} \text{size}:2 \\ \text{userdata}:[1\times 1\ \text{struct}] \end{Bmatrix}$$

1）size 属性（只读）

net.targets{i}.size，该属性定义了第 i 个网络层目标向量中元素的数目，其值为第 i 个网络层神经元的数目（net.layers{i}.size）。

2）userdata 属性

net.targets{i}.userdata，该属性为用户提供了增加关于第 i 个网络层目标向量的用户信息的地方，它预先只定义一个字段，其值为一个提示信息

net.targets{i}.userdata=note:'Put your custom targets information here.'

用户可以通过修改 net.targets{i}.userdata.note 的值，增加关于第 i 个目标向量的用户信息。

7. 网络层权值向量

net.layerWeights={[ ]}

$$\text{net2.layerWeights} = \begin{Bmatrix} [] & [] \\ [1\times 1\ \text{struct}] & [] \end{Bmatrix}$$

每个网络层权值向量的结构属性值可以通过

net.layerWeights{i,j}

访问。例如

$$
\text{net2.layerWeights}\{2,1\} = \begin{bmatrix} \text{delays}: 0 \\ \text{initFcn}: '' \\ \text{learn}: 1 \\ \text{learnFcn}: '\text{learngdm}' \\ \text{learnParam}: [1\times 1 \text{ struct}] \\ \text{size}: [2 \quad 15] \\ \text{userdata}: [1\times 1 \text{ struct}] \\ \text{weightFcn}: '\text{dotprod}' \end{bmatrix}
$$

1）delays 属性

net.layersWeights{i,j}.delays，该属性定义第 j 个网络层与第 i 个网络层权值之间的抽头延迟线。其值为从 0 或正整数开始的逐渐增大的行向量。

2）initFcn 属性

net.layerWeights{i,j}.initFcn，如果网络的初始化函数为 initlay，且网络层的初始化函数为 initwb，则该属性定义从第 j 个网络层到 i 个网络层权值向量的初始化函数。其值可以设置成神经网络工具箱中任意一个权值初始化函数名（权值初始化函数见表 2-10）。

如果网络的初始化函数(net.initFcn)为 initlay，且网络层的初始化函数(net.layers{i}. initFcn)为 initwb，则当 init 函数被调用时，net.layerWeights{i,j}.initFcn 所定义的函数将用于计算从第 j 个网络层到第 i 个网络层的初始权值向量值

net=init(net)

3）learn 属性

net.layerWeights{i,j}.learn，该属性定义第 j 个网络层到第 i 个网络层的权值向量在训练和调整过程中是否改变，其值可设置为 0 或 1，分别表示权值向量在训练和调整过程中禁止和容许进行学习。

[net, Y, E, Pf, Af]=adapt(NET, P, T, Pi, Ai)

[net,tr]=train(NET, P, T, Pi, Ai)

4）learnParam 属性

net.layerWeights{i,j}.learnParam，该属性定义第 i 个网络层来自第 j 个网络层的权值向量学习函数的参数及参数值，其值取决于当前的学习函数（net.layerWeights{i,j}.learnFcn），可以查阅当前学习函数的帮助，获得当前学习函数的具体描述

help(net.layerWeights{i,j}.learnFcn)

5）learnFcn 属性

net.layerWeights{i,j}.learnFcn，如果网络的训练函数为 trainb、trainc 或 trainr，或网络的调整函数为 trains，则该属性定义第 j 个网络层到第 i 个网络层权值向量的学习函数。其值可以设置成神经网络工具箱中任意一个权值学习函数名（权值学习函数如表 2-11 所示）。如果网络的训练函数（net.trainFcn）为 trainb、trainc 或 trainr，或网络的调整函数（net.adaptFcn）为 trains，则当 train 函数被调用时，net.layerWeights{i,j}.learnFcn 所定义的函数将用于计算从第 j 个网络层到第 i 个网络层的权值向量

[net, Y, E, Pf, Af]=adapt(NET, P, T, Pi, Ai)

[net, tr]=train(NET, P, T, Pi, Ai)

6）size 属性（只读）

net.layerWeights{i,j}.size，该属性定义第 i 个网络层来自第 j 个网络层的权值向量元素的数目，其值为具有两个元素的行向量，分别表示相应网络层权值向量（net.LW{i,j}）的行数与列数。第一个元素的值为第 i 个网络层神经元的数目（net.layers{i}.size），第二个元素的值为第 j 个网络层神经元的数目（net.layers{j}.size）与权值抽头延迟线长度的乘积

length(net.layerWeights{i,j}delays)*net.layers{j}.size

7）userdata 属性

net.layerWeights{i,j}.userdata，该属性为用户提供增加关于第 i 个网络层来自第 j 个网络层的权值向量元素数目的用户信息的地方，它预先只定义一个字段，其值为一个提示信息

net.layerWeights{i,j}.userdata=note:'Put your custom weight information here.'

用户可以通过修改 net.layerWeights{i,j}.userdata.note 的值，增加关于第 i 个网络层来自第 j 个输入向量的权值向量元素数目的用户信息。

8）weightFcn 属性

net.layerWeights{i,j}.weightFcn，该属性定义第 i 个网络层来自第 j 个网络层的权值函数。其值可以是神经网络工具箱的任意一个权值函数名（权值函数如表 2-12 所示）。

当 sim 函数被调用时，权值函数被用于网络的仿真

[Y, Pf, Af]=sim(net, P, Pi, Ai)

以上介绍了 MATLAB 神经网络工具箱中神经元和神经网络的一般模型、神经网络对象及其子对象的属性，理解这些内容将有助于我们利用神经网络工具箱进行神经网络的设计和仿真。

通过下面几个例子，读者可以进一步熟悉本章的基本内容。

【例 2-1】设 $\boldsymbol{p}=[2\quad -2]^{\mathrm{T}}$，$\mathrm{IW}^1=\begin{bmatrix}0.5 & 0.5\\ 0.9 & -0.1\\ 0.9 & -0.0\end{bmatrix}$，$\boldsymbol{b}^1=\begin{bmatrix}0.5\\ -0.5\\ 0.4\end{bmatrix}$，$\boldsymbol{f}^1=\begin{bmatrix}\text{purelin}\\ \text{purelin}\\ \text{purelin}\end{bmatrix}$，试画出其网络结构示意图，并求出网络的输出值。

解析：根据题意，神经网络有 1 个输入向量，包含 2 个输入变量；输入层有 3 个神经元，传输函数为 purelin；无其他网络层，所以为单层神经网络。根据以上信息可画出其神经网络结构示意图，如图 2-6 所示。

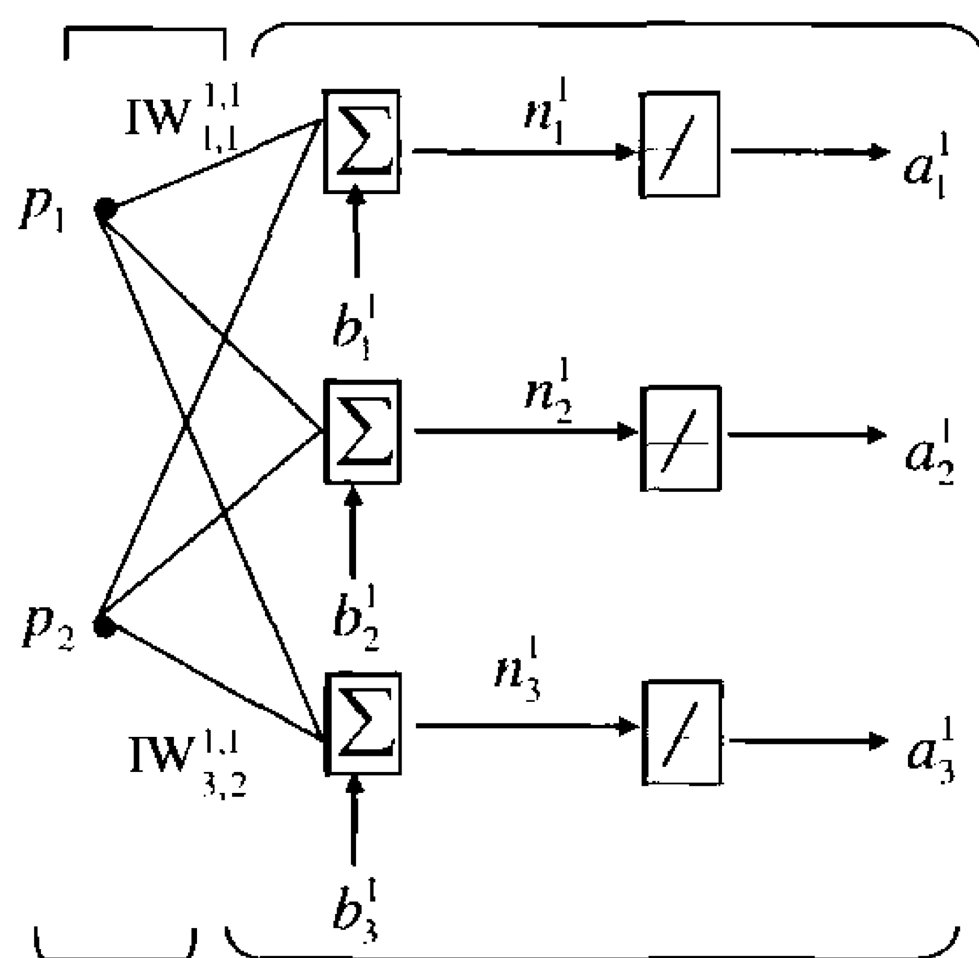


图 2-6　例 2-1 的神经网络的结构示意图

则

$$a^1 = f^1(\text{IW}^1 p + b^1) = f^1 \begin{bmatrix} 0.5 & 0.5 \\ 0.9 & -0.1 \\ 0.9 & 0.1 \end{bmatrix} \begin{bmatrix} 2 \\ -2 \end{bmatrix} + \begin{bmatrix} 0.5 \\ -0.5 \\ 0.4 \end{bmatrix} = \begin{bmatrix} \text{purelin}(0.5) \\ \text{purelin}(1.5) \\ \text{purelin}(2.0) \end{bmatrix} = \begin{bmatrix} 0.5 \\ 1.5 \\ 2.0 \end{bmatrix}$$

【例 2-2】某个神经网络的部分属性如下，据此画出其网络向量模型结构示意图。

```
net.numInputs=1
net.inputs{1}.size=2
net.numLayers=2
net.layers{1}.dimensions=3
net.layers{2}.dimensions=2
net.biasConnect=[1;1]
net.inputConnect=[1;0]
net.layerConnect=[0 0;1 0]
net.outputConnect=[0 1]
net.outputs{1}.size=2
net.layers{1}.transferFcn=logsig
net.layers{2}.transferFcn=logsig
```

解析：根据题意，网络有 1 个输入向量（net.numInputs=1），包含 2 个输入变量（net.inputs{1}.size=2）；有 2 个网络层（net.numLayers=2），第 1 个网络层有 3 个神经元（net.layers{1}.dimensions=3），第 2 个网络层有 2 个神经元（net.layers{2}.dimensions=2），每个网络层的神经元都有阈值（net.biasConnect=[1;1]）；第 1 个网络层（输入层）与输入向量连接，第 2 个网络层与输入向量无连接（net.inputConnect=[1;0]）；只有第 1 个网络层到第 2 个网络层的连接，无其他网络层连接（net.layerConnect=[0 0;1 0]）；两层网络层神经元的传输函数均为 logsig（net.layers{1}.transferFcn=logsig，net.layers{2}.transferFcn=logsig）；第 2 个网络层为输出层，有 2 个输出变量（net.outputConnect=[0 1]，net.outputs{1}.size=2）。据以上所述画出网络向量模型结构示意图，如图 2-7 所示。

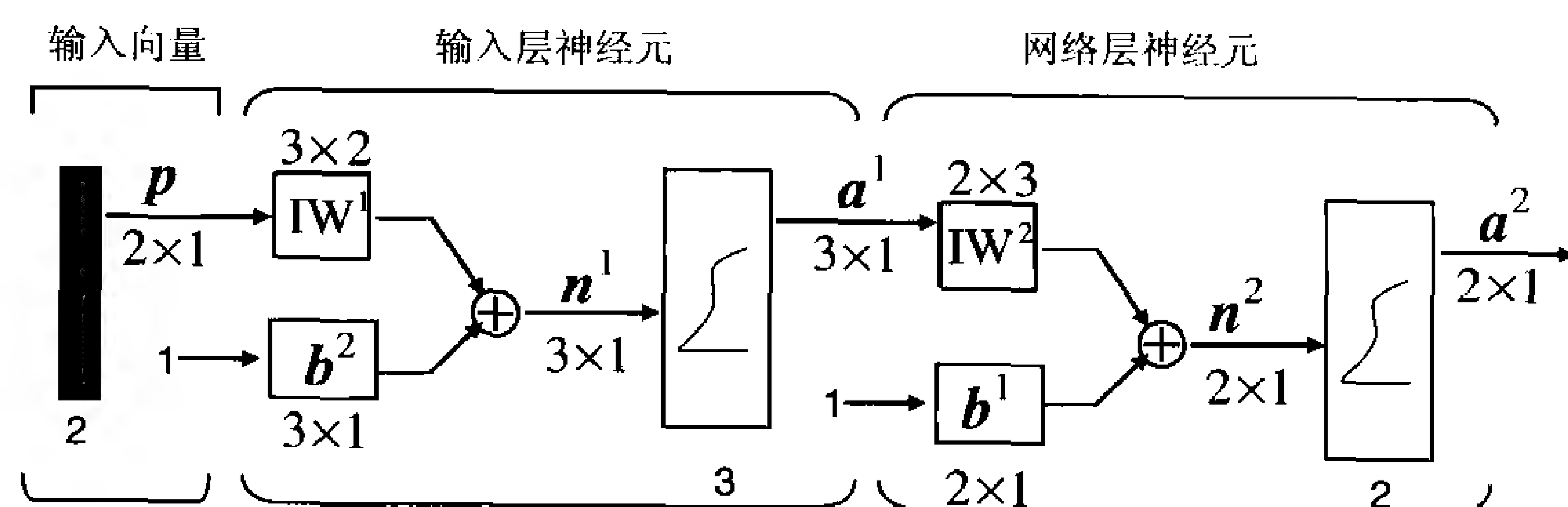


图2-7　例2-1神经网络的向量模型

第 3 章　神经网络工具箱函数的分析及实例

本章将根据神经网络类型的不同及神经网络的发展历程，逐一详细地讲解 MATLAB 与神经网络工具箱函数的使用方法和主要功能。

神经网络工具箱是在 MATLAB 环境下开发出来的许多工具之一。它以人工神经网络理论为基础，利用 MATLAB 编程语言构造出许多典型神经网络的框架和相关的函数。

下面按类型介绍相关函数的方法和功能。

3.1　神经网络的构建函数

表 3-1 列出了神经网络的构建函数，其中 network 函数用于建立由用户自定义的结构特殊的神经网络，这时用户要根据实际情况设定网络的属性参数。一般情况下，用户可利用 newp、newlin 和 newff 等网络建立函数，方便地创建各种常用网络。

表 3-1　构建函数

函数名称	功　　能	函数名称	功　　能
network	构建一个自定义神经网络对象	newlin	构建一个线性神经网络
newc	构建一个竞争层网络	newlind	设计一个线性神经网络
newcf	构建一个前向级联 BP 网络	newlvq	构建一个学习矢量量化网络
newelm	构建一个 Elman 反馈网络	newp	构建一个感知器
newff	构建一个前向 BP 网络	newpnn	设计一个概率神经网络
newfftd	构建一个有输入延迟的前向 BP 网络	newrb	设计一个径向基函数网络
newgrnn	设计一个广义回归网络	newrbe	精确设计一个径向基函数网络
newhop	构建一个 Hopfield 反馈网络	newsom	建立一个自组织映射网络

1．network

在 MATLAB 神经网络工具箱中，使用 network 函数自定义神经网络对象，其调用格式如下。

应用：建立一个自定义神经网络对象。

格式：net=network

net=network(numInputs, numLayers, biasConnect, inputConnect, layerConnect, outputConnect, targetConnect)

解析：该函数返回一个神经网络对象，函数调用形式中的 numInputs 等输入量为确定神经网络结构的参数，其功能说明如下。

numInputs：该属性定义了神经网络的输入个数，属性值可以是 0 或正整数。需要注意的是，该属性定义了网络输入矢量的总个数，而不是单个输入矢量的维数。

numLayers：该属性定义了神经网络的层数，其属性值可以是 0 或正整数。

biasConnect：该属性定义了神经网络的每层是否具有阈值，其属性值为 $N\times1$ 维的布尔量

矩阵，其中 N 为网络的层数（net.numLayers）。net.biasConnect(i)为 1，表示第 i 层上的神经元具有阈值，为 0 则表示该层没有阈值。

inputConnect：该属性定义了神经网络的输入层，其属性值为 $N \times N_i$ 维的布尔量矩阵，其中 N 为网络的层数，N_i 为网络的输入个数（net.numInputs）。net.inputConnect(i,j)为 1，表示第 i 层上的每个神经元都要接收网络的第 j 个输入矢量，为 0 则表示不接收该输入。

layerConnect：该属性定义了网络各层的连接情况，其属性值为 $N \times N$ 维的布尔量矩阵，其中 N 为网络的层数。net.layerConnect(i,j)为 1，表示第 i 层与第 j 层上的神经元相连，为 0 则表示它们不相连。

outputConnect：该属性定义了神经网络的输出层，其属性值为 $1 \times N$ 维的布尔量矩阵，其中 N 为网络的层数。net.outputConnect(i)为 1，表示第 i 层神经元将产生网络的输出，为 0 则表示该层不产生输出。

targetConnect：该属性定义了神经网络的目标层，即网络哪些层的输出具有目标矢量。其属性值为 $1 \times N$ 维的布尔量矩阵，其中 N 为网络的层数。net.targetConnect(i)为 1，表示第 i 层神经元产生的输出具有目标矢量，为 0 则表示该层输出不具有目标矢量。

【例 3-1】定义一个神经网络对象

```
net=network
net =
    Neural Network object:
    architecture:
            numInputs: 0
            numLayers: 0
          biasConnect: []
         inputConnect: []
         layerConnect: []
        outputConnect: []
        targetConnect: []
           numOutputs: 0    (read-only)
           numTargets: 0    (read-only)
    numInputDelays: 0    (read-only)
    numLayerDelays: 0    (read-only)
    subobject structures:
               inputs: {0x1 cell} of inputs
               layers: {0x1 cell} of layers
              outputs: {1x0 cell} containing no outputs
              targets: {1x0 cell} containing no targets
               biases: {0x1 cell} containing no biases
         inputWeights: {0x0 cell} containing no input weights
         layerWeights: {0x0 cell} containing no layer weights
    functions:
             adaptFcn: (none)
              initFcn: (none)
           performFcn: (none)
             trainFcn: (none)
    parameters:
```

```
        adaptParam: (none)
         initParam: (none)
      performParam: (none)
        trainParam: (none)
    weight and bias values:
                        IW: {0x0 cell} containing no input weight matrices
                        LW: {0x0 cell} containing no layer weight matrices
                         b: {0x1 cell} containing no bias vectors
    other:
            userdata: (user stuff)
```

上例在调用函数 network 时没有给出网络的结构参数，因此函数返回值 net 中的各属性参数均为函数提供的默认值。

2．newc

应用：建立一个竞争层网络。

格式：net=newc

net=newc(PR, S, KLR, CLR)

解析：竞争层网络由单一的竞争层构成，主要用于解决分类问题。竞争层的加权函数为 negdist，输入函数为 netsum，传递函数为 compet。神经元权值和阈值的初始化函数分别为 midpoint 和 initcon，网络的自适应调整函数和训练函数分别为 trains 和 trainr，权值和阈值的学习函数分别为 learnk 和 learncon。其参数说明如下。

PR：网络输入矢量取值范围的矩阵[Pmin Pmax]；

S：竞争层神经元的个数，也是网络输入矢量的分类数；

KLR：权值学习速率，默认值为 0.01；

CLR：阈值学习速率，默认值为 0.001。

【例 3-2】利用竞争层网络把下列二维矢量分为两类。

```
P=[0.1 0.8 0.1 0.9 0.2 0.8;
    0.2 0.9 0.1 0.8 0.1 0.8];
```

该二维输入矢量的取值范围矩阵为

```
PR=[0 1:0 1];
```

若把这些矢量分为两类，新建的竞争层应由两个神经元构成

```
net=newc([0 1;0 1],2);
```

对网络进行训练，则语句调用格式为

```
net=train(net,P);
```

网络的仿真结果为

```
Y=sim(net,P)
Y =
    (1,1)          1
    (2,2)          1
    (1,3)          1
    (2,4)          1
    (1,5)          1
```

```
(2,6)          1
```

其中，***Y*** 是一个稀疏矩阵，其第一行的 2、4、6 列元素和第二行的 1、3、5 列元素均为 1，即 ***P*** 中第 2、4、6 个输入矢量为第一类，其余三个矢量属于第二类。为了便于观察结果，可将 ***Y*** 转化为下标矢量形式，即

```
Yc=vec2ind(Y)
Yc =
     1     2     1     2     1     2
```

3. newcf

应用：构建一个前向级联 BP 网络。

格式：net=newcf

net=newcf(PR, [S1,S2, …, SN]，{TF1 TF2…TFN1},BTF, BLF,PF)

解析：net=newcf 用于在对话框中创建一个 BP 网络。其参数说明如下。

PR：由每组输入（共有 *R* 组输入）元素的最大值和最小值组成的 $R\times2$ 维的矩阵；

S*i*：第 *i* 层的长度，共计 N_1 层；

TF*i*：第 *i* 层的传递函数，默认为“tansig”；

BTF：BP 网络的训练函数，默认为“trainlm”；

BLF：权值和阈值的 BP 学习算法，默认为“learngdm”；

PF：网络的性能函数，默认为“mse”。

【例 3-3】假设输入和目标矢量矩阵分别为

```
P=[8 7 6 5 4 3 2 1 0];
T=[0 1 2 3 2 1 2 3 2];
```

建立一个两层前向级联网络，其中第一层有六个神经元，采用 tansig 传递函数，输出层神经元的传递函数为 purelin，其他参数均采用默认值。

```
net=newcf([0 8],[6 1],{'tansig' 'purelin'});
```

对网络进行训练和仿真

```
net=train(net,P,T);
Y=sim(net,P)
```

训练和仿真结果为

```
Y =
 0.0000    1.0000    2.0000    3.0000    2.0000    1.0000   2.0000   3.0000     2.0000
```

4. newelm

应用：构建一个 Elman 反馈网络。

格式：net=newelm

net=newelm(PR, [S1,S2, …, SN],{TF1 TF2…TFN1},BTF, BLF,PF)

解析：该函数可以构建一个 *N* 层的 Elman 网络。网络中各层依次级联，除最后一层外，网络的每层都具有自反馈，最后一层为网络的输出层。网络各层的加权函数为 dotprod，输入函数为 netsum，各层传递函数由用户设定。各神经元权值和阈值的初始化函数为 initnw，网络的自适应调整函数为 trains，并根据指定的学习函数对权值和阈值进行更新，网络的训练函数由用户指定。各参数说明见 newcf。

【例 3-4】假设输入和目标序列分别为

```
Pseq={0,0,1,1,0,1,0,0,1};
Tseq={0,1,0,1,0,0,0,1,0};
```

构建一个两层 Elman 网络，其中第一层有六个神经元，传递函数为 tansig，该层具有自反馈，输出层神经元的传递函数为 logsig，其他参数均采用默认值。

```
net=newelm([0,1],[6,1],{'tansig' 'logsig'})
```

对网络进行训练和仿真

```
net.trainParam.epochs=500;
net=train(net,Pseq,Tseq);
Y=sim(net,Pseq)
```

训练和仿真结果为

```
Y =
[0.0355] [0.9606] [0.0389][0.9277] [0.0174] [0.0301] [0.0416] [0.9796] [0.0346]
```

5．newff

应用：构建一个前向 BP 网络。

格式：net=newff(PR, [S1,S2, …, SN],{TF1 TF2…TFN1},BTF, BLF,PF)

解析：在输入参数中，PR 为 R 维的输入元素的 $R\times2$ 最大最小值矩阵；Si 为第 i 层网络神经元的个数，共有 N_1 层；TFi 为第 i 层网络的转移函数，默认为 tansig 函数；BTF 为神经网络的训练函数，默认为 trainlm 函数；BLF 为神经网络权值/偏差的学习函数；PF 为性能评价函数，默认为 mse 函数。

【例 3-5】对以下样本数据，采用 newff 构建前向神经网络。

```
P=[8 7 6 5 4 3 2 1 0];
T=[0 1 2 3 2 1 2 3 2];
net=newff([0 8],[6 1],{'tansig' 'purelin'});
```

对网络进行训练和仿真

```
net=train(net,P,T);
Y=sim(net,P)
```

训练和仿真结果为

```
Y =
  0.0031   0.9914   2.0086   2.9969   2.0000   0.9999   1.9998   2.9998   1.9998
```

6．newfftd

应用：构建一个具有输入延迟的前向 BP 网络。

格式：net=newfftd(PR,ID, [S1,S2, …, SN],{TF1 TF2…TFN1},BTF, BLF,PF)

解析：该函数用于建立一个 N 层前向 BP 网络，网络输入经由指定延迟后进入网络的第一层。网络各层的加权函数为 dotprod，输入函数为 netsum，各层传递函数由用户设定。各神经元权值和阈值进行更新，网络的训练函数由用户指定。ID 为输入延迟矢量，其他参数说明参见上面的构建函数。

【例 3-6】假设输入和目标序列分别为

```
P={1 0 0 1 1 0 1 0};
```

```
T={1 -1 0 1 0 -1 1 -1};
```

构建一个两层网络，网络的第一层有三个神经元，该层输入由网络输入及其一拍延迟量构成，神经元传递函数为 tansig，输出层神经元传递函数为 purelin。

```
net=newfftd([0 1],[0 1],[3 1],{'tansig' 'purelin'});
```

对网络进行训练和仿真

```
net.trainParam.epochs=60;
net=train(net,P,T);
Y=sim(net,P)
```

输出结果为

```
Y =
 [1.0000] [-1.0000] [-5.0460e-013] [1.0000][-2.1982e-012] [-1.0000 [1.0000]   [-1.0000]
```

7. newgrnn

应用：设计一个广义回归网络。

格式：net=newgrnn

net=newgrnn(P, T, spread)

解析：广义回归网络是径向基函数网络的变形，主要用于解决函数逼近问题，本函数可以快速设计一个广义回归网络。广义回归网络是一个两层神经网络，第一层采用径向基神经元，传递函数为 radbas，加权函数为 dist，输入函数为 netprod；第二层神经元的传递函数为 purelin，加权函数为 normprod，输入函数为 netsum。其参数说明如下。

P：网络输入样本矢量构成的矩阵；

T：网络输出目标矢量构成的矩阵；

spread：扩展常数，spread 越大，则函数越平滑，当 spread 小于输入矢量间的典型距离时，网络拟合的函数将非常逼近于样本矢量数据，spread 的默认值为 1。

【例 3-7】假设输入和目标矢量的矩阵分别为

```
P=[0 1 2 3 4];
T=[-1 -3.1 -5.4 -4.3 -2];
```

设计一个广义回归网络，其仿真如下。

```
net=newgrnn(P,T);
Y=sim(net,P)
```

仿真结果为

```
Y =
   -1.8511    -3.1837    -4.3706    -3.9699    -2.8723
```

8. newlin

应用：新建一个线性神经网络函数。

格式：net=newlin

net=newlin(PR, S, ID, LR)

解析：net=newlin 返回一个没有定义结构的空对象，并显示图形用户界面函数 nntool 的帮助文字；net=newlin(PR, S, ID, LR)中 net 为生成的线性神经网络。其参数解析如下。

RP：网络输入向量中的最小值和最大值组成的矩阵[Pmin, Pmax]；

S：输出向量的个数；
ID：输入延时向量（可省略）；
LR：学习速率（可省略），默认值为 0.01。
【例 3-8】假设输入和目标矢量矩阵为

```
P=[0 1 2 3 4];
T=[-1 -3.1 -5.4 -7.3 -8.2];
```

建立一个单神经元线性网络。

```
net=newlin([0 5],1);
```

对网络进行训练和仿真

```
net=train(net,P,T);
Y=sim(net,P)
```

输出结果为

```
Y =
    -1.1557    -3.0593    -4.9629    -6.8665    -8.7701
```

9. newlind

应用： 设计一个线性层。
格式： net=newlind(P,T,Pi)
解析： 其中 P、T 分别是训练样本的输入矩阵和目标输出向量，Pi 是初始输入延时 cell 向量。
【例 3-9】使用 newlind 函数构建线性神经网络逼近层。

$$y=\begin{cases}\sin(2\pi t) & 0\leqslant t\leqslant 2\\ \sin(4\pi t) & 2\leqslant t\leqslant 4\\ \sin(8\pi t) & 4\leqslant t\leqslant 6\end{cases}$$

首先，产生输入训练样本和训练目标样本。

```
time1=0:0.01:2;
time2=2.01:0.01:4;
time3=4.01:0.01:6;
time=[time1 time2 time3];
T=[cos(time1*2*pi) cos(time2*4*pi) cos(time3*8*pi)];
Q=length(T);
P=zeros(6,Q);
P(1,2:Q)=T(1,1:(Q-1));
P(2,3:Q)=T(1,1:(Q-2));
P(3,4:Q)=T(1,1:(Q-3));
P(4,5:Q)=T(1,1:(Q-4));
P(5,6:Q)=T(1,1:(Q-5));
P(6,7:Q)=T(1,1:(Q-6));
```

然后使用 newlind 函数设计线性层。

```
net=newlind(P,T);
```

```
a=sim(net,P);
plot(time,T,time,a,'k-o');
legend('给定输入信号','网络输出信号')
figure
plot(time,a-T)
title('误差曲线')
```

线性层网络函数逼近效果和训练误差曲线如图 3-1 和 3-2 所示。

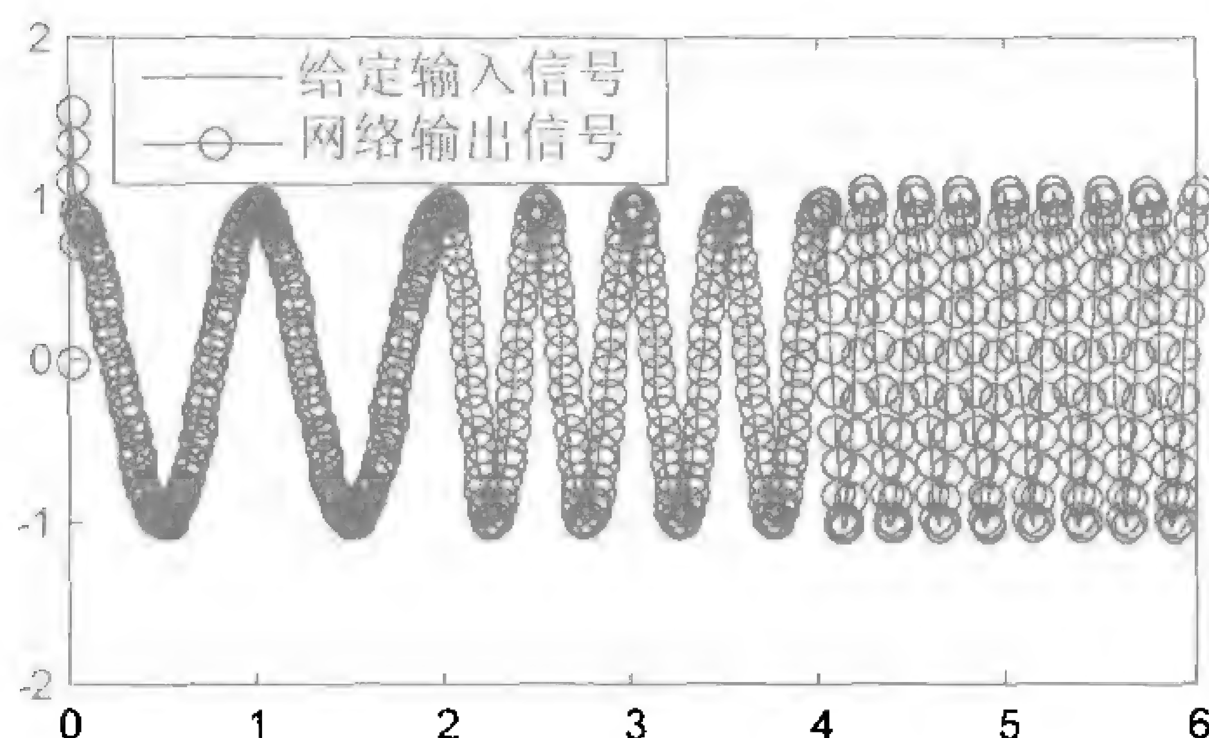
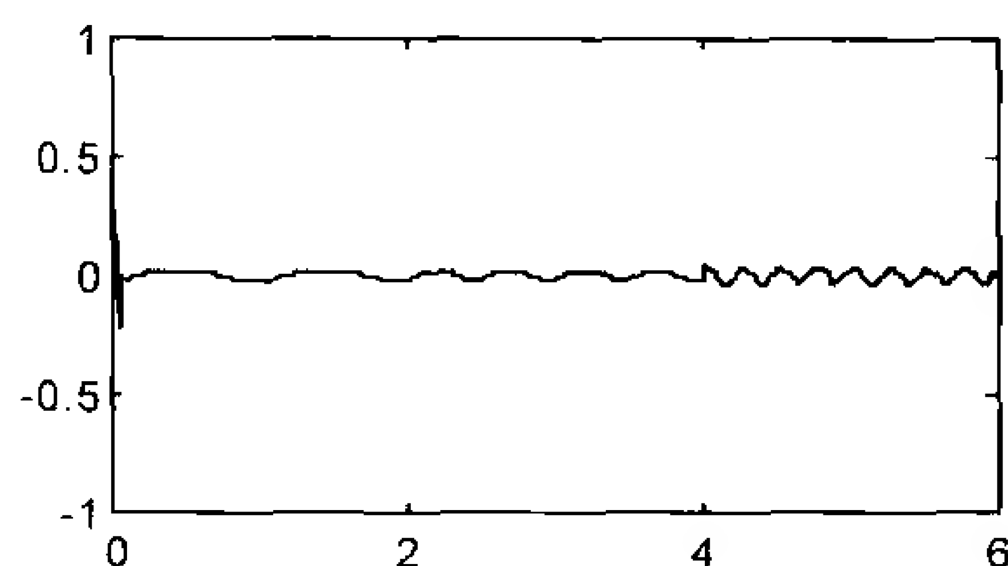


图 3-1　newlind 的函数逼近效果图


图 3-2　训练误差曲线效果图

10．newlvq

应用：建立一个学习矢量量化网络。

格式：net=newlvq

net=newlvq(PR, S1, PC, LR, LF)

解析：学习矢量量化网络主要用于解决分类问题。该网络由两层神经元组成，网络的第一层为竞争层；第二层中的每个神经元只与竞争层中的部分神经元相连，网络中的每个神经元都没有阈值。网络第一层神经元的传递函数为 compet，加权函数为 negdist，权值初始化函数为 midpoint；网络第二层神经元的传递函数为 purelin，加权函数为 dotprod，权值都为 1。网络使用 trains 和 trainr 函数进行自适应调整和训练，这两个函数使用指定的学习函数对竞争层神经元的权值进行修正。其参数说明如下。

PR：网络输入矢量取值范围的矩阵[Pmin, Pmax]；

S1：竞争层神经元的个数；

PC：s^2（直接由 PC 的维数确定）维矢量，其中 s^2 为网络输入矢量的类别数，也是输出层的神经元个数；PC(i)为网络输入矢量中第 i 类矢量所占的比例，即输出层中和竞争层第 i 个神经元相连的神经元所占的比例；

LR：学习速率，默认值为 0.01；

LF：学习函数，可以是 learnlv1 或 learnlv2 函数，默认值为'learnlv1'，网络只有先使用 learnlv1 函数进行学习后，才能再用 learnlv2 进行学习。

【例 3-10】假定要将下列样本矢量分为两类，每类所占的比例均为 0.5。

```
P=[3 -2 -1 2 2 -1;
   2 -1 -1 2 3 -2];
T=[1 2 2 1 1 2];
```

建立学习矢量量化网络，网络竞争层由四个神经元组成。

```
net=newlvq([-2 3;-2 3],4,[0.5 0.5]);
```

下标矢量 T 必须转化为单值矢量组，才能作为目标矢量使用

```
T1=ind2vec(T);
net=train(net,P,T1);
```

仿真结果为

```
Y=vec2ind(sim(net,P))
Y =
     1     2     2     1     1     2
```

11．newhop

应用： 该函数用于设计一个 Hopfield 网络。

格式： net=newhop

net=newhop(T)

解析： net=newhop 用于在对话框中创建一个 Hopfield 网络；

T：Q 个目标向量组成的 $R \times Q$ 维矩阵，矩阵元素必须为 1 或-1；

net：函数返回值，创建的 Hopfield 网络稳定点位于目标向量 T 中。

【例 3-11】创建一个 Hopfield 网络并进行仿真。

```
T=[-1 -1 1;1 -1 1]';
%创建一个 Hopfield 网络
net=newhop(T);
Ai={[-0.9;-0.8;0.7]};
[Y,Pf,Af]=sim(net,{1 5},{},Ai);
Y{1}'
```

结果为

```
ans =
    -1    -1     1
```

由于 Y{1}=T(:, 1)，可以看出，设计的 Hopfield 网络已经成功地将存在偏差的向量 Ai 转换为最接近的目标向量 $\boldsymbol{T}$。

12．newp

应用： 构造感知器模型。

格式： net=newp(PR, S, TF, LF)

解析： PR：$R \times 2$ 的输入向量最大值和最小值构成的矩阵；

S：神经元的个数；

TF：传输函数设置，为 hardlim 函数或者 hardlins 函数，默认为 hardlim 函数；

LF：学习函数设置，为 learnp 函数或者 learnpn 函数，默认为 learnp 函数；

net：生成的感知器网络。

【例 3-12】生成具有 1 个神经元的神经网络。

```
%输入向量
p=[1.24 1.36 1.38 1.38 1.38 1.4 1.48 1.54 1.56 1.14 1.18 1.2 1.26 1.28 1.3;
   1.72 1.74 1.64 1.82 1.9 1.7 1.82 1.82 2.08 1.78 1.96 1.86 2.0 2.9 1.96];
%目标向量
T=[1 1 1 1 1 1 1 1 1 0 0 0 0 0 0];
net=newp([0 3;0 3],1);
```

查看工作窗口中的变量，可以得到

```
>> whos
  Name        Size                    Bytes  Class
  T           1x15                      120  double array
  net         1x1                     18543  network object
  p           2x15                      240  double array
Grand total is 676 elements using 18903 bytes
```

net 的变量属性是网络对象的结构体，可以输入 net 后查看 net 的结构体的相关信息。

```
>> inputweigths=net.inputweights{1}
inputweigths =
          delays: 0
         initFcn: 'initzero'
           learn: 1
        learnFcn: 'learnp'
      learnParam: []
            size: [1 2]
        userdata: [1x1 struct]
       weightFcn: 'dotprod'
```

通过以下命令查看神经网络各层权重以及神经元阈值情况。

```
>> net.IW{1}   %输入层神经元权重
ans =
     0     0
>> net.LW{1}   %隐层神经元权重
ans =
     []
>> net.b{1}   %神经元阈值
ans =
     0
```

13. newrb

应用：构建径向基网络。

格式：[net, tr]=newrb(P, T, goal, spread, MN, DF)

解析：在径向基网络设计中，spread 参数对径向基网络的性能影响较大，通常情况下，spread 的值较大时，径向基网络逼近曲线越光滑，当 spread 值过小时，径向基函数的逼近效果就会变差。其参数说明如下。

P、T：分别为训练样本输入和目标输出；

goal：为径向基网络输出的总平均误差方差；

spread：为径向基函数 radbas 的密度常数；

MN：为最大的神经数目。

【例 3-13】创建一个径向基网络。

```
P=[1 2 3 4 5];
T=[0.5 2.3 3.2 4.6 6.7];
err_goal=0.01
spr_cons=1;    %径向基函数密度常数
net=newrb(P,T,err_goal,spr_cons);
```

```
A=sim(net,P)          %回代检验
postreg(A,T)
```

径向基网络输出和训练目标输出的回归曲线如图 3-3 所示。回代检验的输出结果为

```
A =
    0.5000    2.3000    3.2000    4.6000    6.7000
```

对新样本进行训练，输入以下程序段

```
>> p=3.5;
>> a=sim(net,p)
```

输出结果为

```
a =
    3.5678
```

newrb 函数径向基网络输出回归曲线如图 3-3 所示。

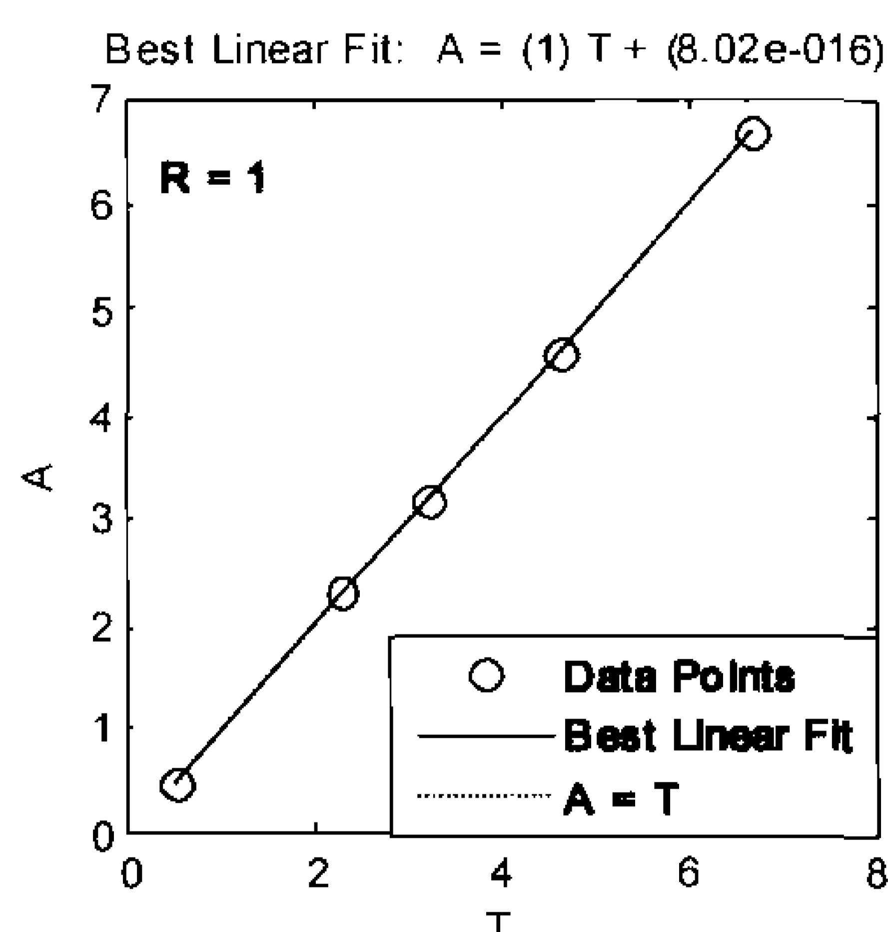


图 3-3　newrb 函数径向基网络输出回归曲线

14．newrbe

应用：创建一个准确的径向基网络，可以实现网络输出与目标输出零误差。

格式：net= newrbe(P,T,spread)

newrbe 函数参数的意义与 newrb 完全相同，具体介绍见 newrb 函数。

【例 3-14】创建一个准确的径向基函数。

```
P=[1 2 3 4 5];
T=[0.5 2.3 3.2 4.6 6.7];
spr_cons=1;
net=newrbe(P,T,spr_cons);
A=sim(net,P)
postreg(A,T)
p=3.5;
a=sim(net,p)
```

该程序段首先使用 newrbe 函数创建一个准确的径向基函数，然后使用训练样本数据回代检验，并绘制网络输出的回归曲线，最后给出测试样本的输出。newrbe 函数准确径向基网络输出同目标矢量的回归曲线如图 3-4 所示。可以发现此时 newrbe 函数创建的准确径向基网络能够实现目标矢量零误差输出。回代检验结果和测试样本输出结果如下

```
A =
```

```
    0.5000    2.3000    3.2000    4.6000    6.7000
a =
    3.6339    %同 newrb 函数创建的径向基网络输出有差异
```

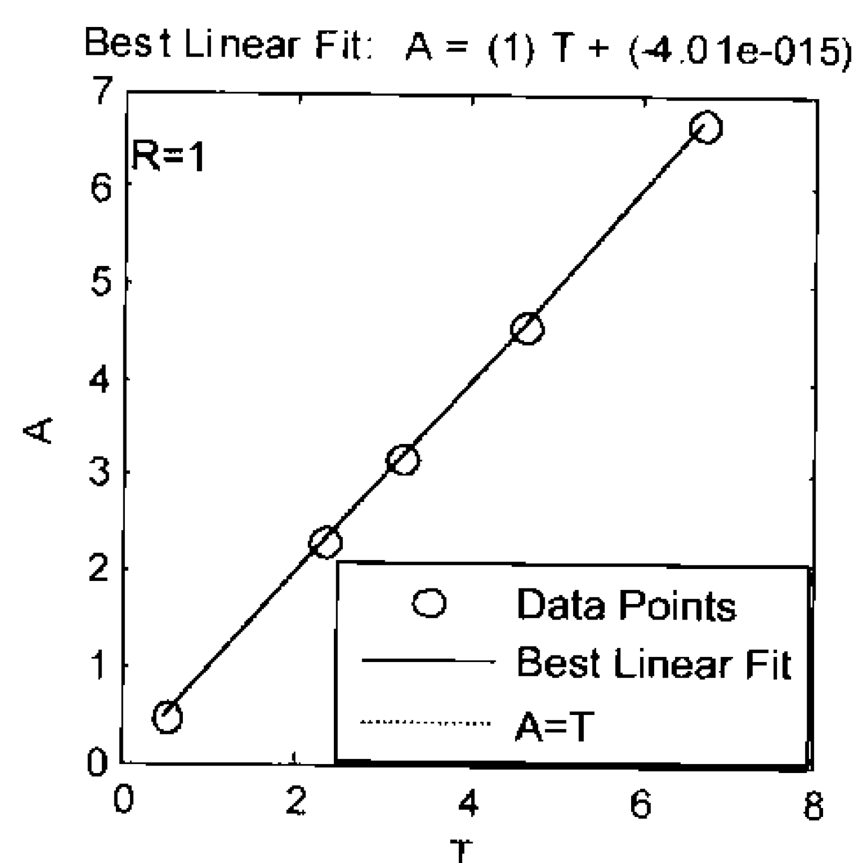


图 3-4　newrbe 函数准确径向基网络输出同目标向量的回归曲线

15. newpnn

应用：创建一个概率神经网络。

格式：net=newpnn(P,T,spread)

解析：newpp 函数的参数意义参见 newrb 函数相关介绍。下面通过实例演示 newpnn 函数的使用。

【例 3-15】创建一个概率神经网络。

```
P=[0 0 2 3 2 2 1;0 2 1 2 1 3 1];
T=[1 1 2 1 2 1 2];
Tc=ind2vec(T)
```

ind2vec 函数将下标变换成单值向量

```
Tc =
    (1,1)        1
    (1,2)        1
    (2,3)        1
    (1,4)        1
    (2,5)        1
    (1,6)        1
    (2,7)        1
```

创建概率神经网络，并对输入样本进行训练

```
>> net=newpnn(P,Tc);     %创建概率神经网络
Ac=sim(net,P)
A=vec2ind(Ac)            %将单值向量转化为下标形式
```

输出结果为

```
A =
     1     1     2     1     2     1     2
```

从输出结果可以看出，回代误差为 0，对输入向量进行了正确的分类。对新的样本进行训练

```
>> p=[2 3;2 1]
a=vec2ind(sim(net,p))
```

返回结果为

```
a =
     1     2
```

16．newsom

应用： 建立一个自组织映射网络。

格式： net=newsom

net=newsom(PR, [D1,D2…], TFCN, DFCN, OLR, OSTEPS, TLR, TND)

解析： net=newsom 表示在对话框中创建一个新的网络；

PR：R 个输入元素的最大值和最小值设定值，$R\times2$ 维矩阵；

Di：第 i 层的维数，默认为[5，8]；

TFCN：拓扑函数（即结构函数），默认为"hextop"；

DFCN：距离函数，默认为"linkdist"；

OLR：分类阶段学习速率，默认为 0.9；

OSTEPS：分类阶段的步长，默认为 1000；

TLR：调谐阶段的学习速率，默认为 0.02；

TND：调谐阶段的领域距离，默认为 1。

【例 3-16】假定要把下列样本矢量分为四类

```
P=[3 -2 1 2 -2 -1 -2 1;
   2 -1 -1 2 3 -2 2 3];
```

则需要建立一个四神经元的自组织映射网络。

```
net=newsom([-2 3;-2 3],[2 2]);
```

网络训练和仿真结果为

```
net=train(net,P);
Y=vec2ind(sim(net,P))
Y =
     4     1     3     4     2     1     2     4
```

3.2　神经网络的应用函数

表 3-2 列出了神经网络的基本应用函数，它们可分别用来对神经网络进行训练、显示、初始化和仿真，调用这些函数可以完成对神经网络对象的基本操作。

表 3-2　神经网络的应用函数

函数名称	功　能	函数名称	功　能
adapt	对神经网络进行自适应调整	initlay	对神经网络逐层进行初始化
disp	显示神经网络对象的属性	revert	对神经网络的权值和阈值重新设置为初始值
display	显示神经网络对象的名称和属性		
init	对神经网络的参数进行初始化	train	对神经网络进行训练
initnw	待初始化的神经网络	sim	对神经网络进行仿真
initwb	层初始化函数		

1．adapt

应用：对神经网络进行自适应训练。

格式：[net, Y, E, Pf, Af, tr]=adapt(net, P,T, Pi, Ai)

解析：输入参数说明如下。

P：训练样本数据，对于 n 组训练样本，R 个输入端，则 P 的数据格式为 $R\times n$ 的数组。

T：目标样本数据，相对应于 P 数据，则为 $1\times n$ 的输出向量。

Pi：初始化输入层延迟条件。

Ai：初始化层延迟条件。

输出参数说明如下。

Y：神经网络的输出数据，数据大小格式同输入的目标样本数据，为 $1\times n$ 的向量。

E：神经网络的输出误差数据，大小为 $1\times n$ 的向量，使用 sse 函数或者 mse 函数计算网络误差。

Pf：训练后的网络输出层延迟条件。

Af：训练后的网络层延迟条件。

tr：网络训练过程数据记录，包括 epochs 和 pref 数据。

对例 3-12 构建的感知器网络进行自适应训练，得到以下结果

```
>> [net,Y,E,Pf,Af,tr]=adapt(net,p,T);
>> Y
Y =
    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
>> E
E =
    0    0    0    0    0    0    0    0    0   -1   -1   -1   -1   -1   -1
>> tr
tr =
     timesteps: 1
          perf: 0.4000
>> mse(E)
ans =
     0.4000
>> sse(E)
ans =
       6
```

2．disp 和 display

应用：显示神经网络对象的属性。

格式：disp(net)

display(net)

解析：这两个函数都用于显示神经网络对象的属性，唯一不同之处在于 display 函数还要显示神经网络对象的名称，而 disp 则不显示。

3．init

应用：对神经网络的参数进行初始化。

格式：net=init(net)

解析：该函数通过调用初始化函数 net.initFcn，并根据初始化函数参数 net.initParam 对网络权值和阈值进行初始化。

4．initlay

应用：对每一层的结构神经网络进行初始化。

格式：NET=initlay(net)

info=initlay(code)

解析：net：待初始化的神经网络；

NET：初始化后的神经网络；

info=initlay(code)：根据不同的 code 代码返回不同的信息，包括如下。

pnames：初始化参数的名称。

pdefaults：默认的初始化参数。

通过指定神经网络每一层 *i* 的初始化函数 NET.layers{i}来调用该函数，初始化后的神经网络的每一层都得到了修正。

5：initnw

应用：对层神经网络进行初始化。

格式：NET=initnw(net, i)

解析：net：待初始化的神经网络；

i：层次索引；

NET：初始化后的神经网络。

6．initwb

应用：该函数也是一个层的初始化函数，它按照设定的每层的初始化函数对每层的权值和阈值进行初始化。

格式：NET=initwb(net, i)

解析：net：待初始化的神经网络；

i：层次索引；

NET：初始化后的神经网络。

【例 3-17】根据给定的输入向量 ***P*** 和目标向量 ***T***，创建一个感知器网络，对其进行训练并初始化。

```
P=[0 1 0 1;0 0 1 1];
T=[1 0 0 0];
net=newp([0 1;-2 2],1);   %创建一个感知器网络
%感知器的 P 和阈值
net.iw{1,1}
net.b{1}
%对感知器进行训练
net=train(net,P,T);
net.iw{1,1}
net.b{1}
net=init(net);
net.iw{1,1}
net.b{1}
```

输出结果为：

```
ans =
      0      0
ans =
      0
TRAINC, Epoch 0/100
TRAINC, Epoch 4/100
TRAINC, Performance goal met.
ans =
     -1     -1
ans =
      0
ans =
      0      0
ans =
      0
```

可见，在感知器刚刚创建，尚未进行训练以前，权值和阈值都为 0；训练以后，权值和阈值分别为[1 2]和−3；对训练好的网络进行初始化后，恢复到以前的值，都为 0。

7．revert

应用：该函数将网络中的权值和阈值恢复为初始值。

格式：net=revert(net)

解析：该函数将网络中的权值和阈值恢复为最近一次初始化时产生的初始数值，如果网络结构在初始化后发生了变化，权值和阈值将不能恢复，这时将把它们都设置为 0。

8．train

应用：用于对神经网络进行训练。

格式：[net, tr, Y, E, Pf, Af]=train(NET, P, T, Pi, Ai)

[net, tr, Y, E, Pf, Af]= train(NET, P, T, Pi, Ai, VV, TV)

解析：输入参数说明如下。

NET：待训练的神经网络；

P：网络的输入信号；

T：网络的目标，默认为 0；

Pi：初始的输入延迟，默认为 0；

Ai：初始的层次延迟，默认为 0；

VV：网络结构确认向量，默认为空；

TV：网络结构测试向量，默认为空。

输出参数说明如下。

net：函数返回值，训练后的神经网络；

tr：函数返回值，训练记录（包括步数和性能）；

Y：函数返回值，神经网络输出信号；

E：函数返回值，神经网络的误差；

Pf：函数返回值，最终输入延迟；

Af：函数返回值，最终层延迟。

【例 3-18】使用 train 函数训练感知器网络。

首先，输入训练样本和训练目标样本，创建单神经元感知器网络，设置训练函数 train 函数的参数，并进行训练，训练过程曲线如图 3-5 所示。

```
%输入向量
P=[1.24 1.36 1.38 1.38 1.38 1.4 1.48 1.54 1.56 1.14 1.18 1.2 1.26 1.28 1.3;
   1.72 1.74 1.64 1.82 1.9 1.7 1.82 1.82 2.08 1.78 1.96 1.86 2.0 2.9 1.96];
%目标向量
T=[1 1 1 1 1 1 1 1 1 0 0 0 0 0 0];
net=newp([0 3;0 3],1);
%训练函数 train 的参数设置
net.trainParam.epochs=500;
net.trainParam.goal=0;
net.trainParam.show=50;
%使用 train 函数训练单神经元感知器网络
[net,tr,Y,E,Pf,Af]=train(net,P,T);
%使用测试样本数据并绘制分类图
figure
p=[1.24,1.28,1.4,;1.8,1.84,2.04];
a=sim(net,p);
plotpv(p,a);
point=findobj(gca,'type','line');
set(point,'Color','red');
hold on;
plotpv(P,T);
plotpc(net.IW{1},net.b{1});
```

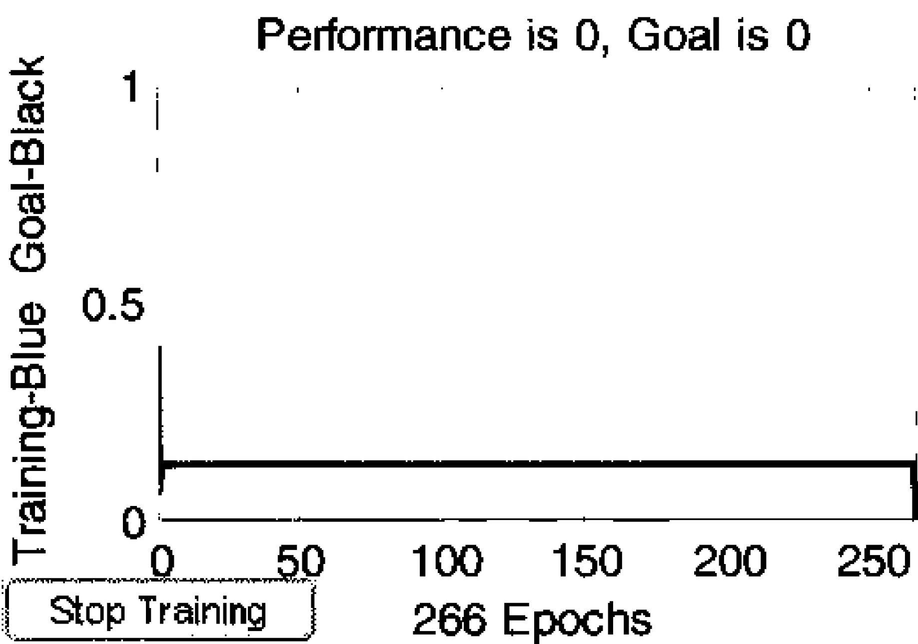


图 3-5　单神经元感知器训练过程曲线

然后利用训练好的单神经元感知器模型仿真检验样本数据，结果如图 3-6 所示。

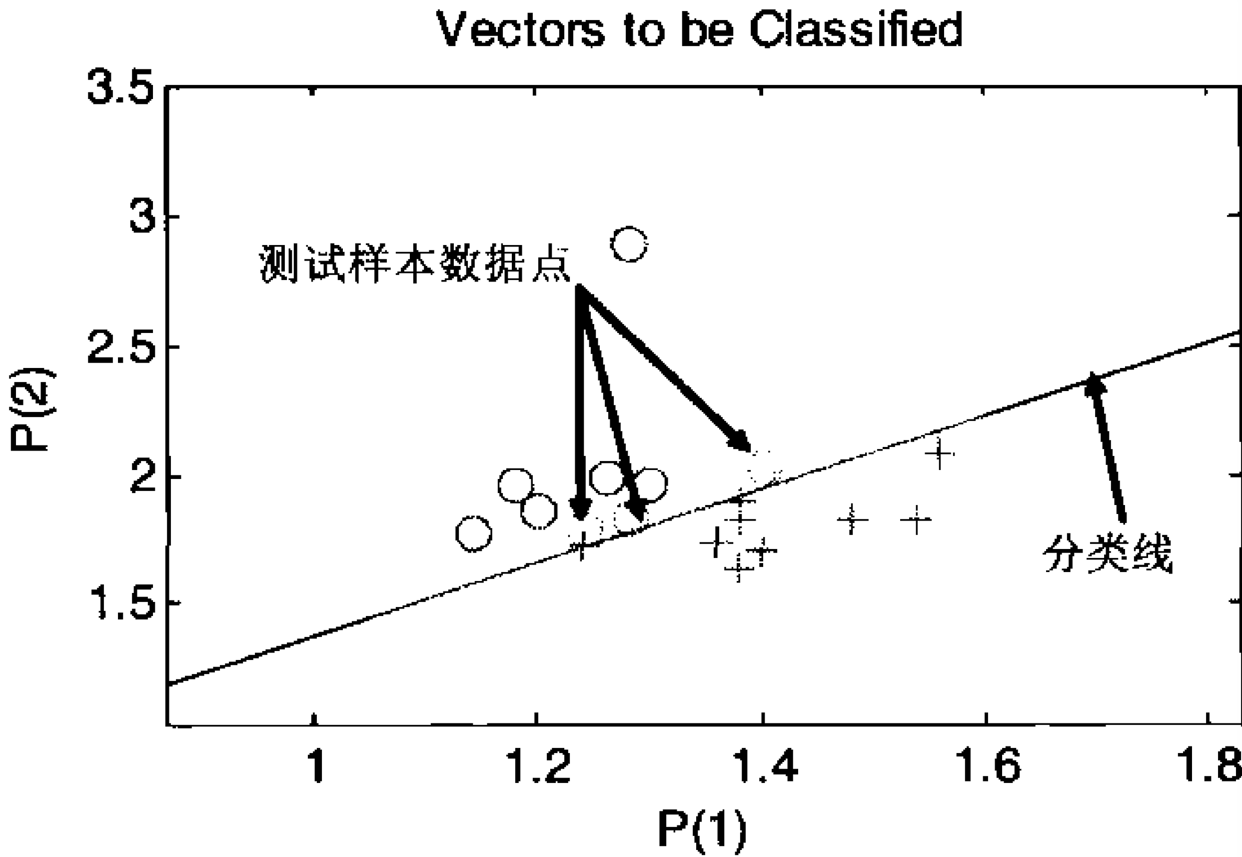


图 3-6　测试样本训练结果

9．sim

应用：对神经网络进行仿真。

格式：[Y, Pf, Af, E, pref]=sim(net, P, Pi, Ai, T)

[Y, Pf, Af, E, pref]=sim(net, {Q TS}, Pi, Ai, T)

[Y, Pf, Af, E, pref]=sim(net, Q, Pi, Ai, T)

解析：输入/输出参数的设置和说明可以参考 adapt 函数。使用 sim 函数同样可以实现感知器网络的训练。利用上文中建立的感知器神经元模型，训练新的样本数据。

```
p=[1.24,1.28,1.4;1.8,1.84,2.04];
a=sim(net,p);
```

输出结果：

```
a =
      1      1      1
```

3.3　权值和阈值初始化函数

权值和阈值初始化函数如表 3-3 所示。

表 3-3　权值和阈值初始化函数

函数名称	功　能	函数名称	功　能
initcon	对阈值进行公平初始化	randnc	对权值进行列归一化初始化
initzero	把权值和阈值初始化为零	randnr	对权值进行行归一化初始化
midpoint	把权值初始化为输入矢量值域的中心	rands	对权值和阈值进行随机初始化

网络属性设置：

如果要设计网络第 j 层的权值和阈值初始化函数，可按如下步骤进行：

- 将网络初始化函数 net.initFcn 设置为'initlay'；
- 将该层的初始化函数 net.layers{i}.initFcn 设置为'initwb'；
- 将该层的输入权值、网络权值和阈值的初始化函数 net.inputWeights{i,j}.initFcn、net.layerWeights{i,j}.initFcn 和 net.biases{i}.initFcn 分别设置为期望的初始化函数。

1．initcon

应用：对阈值进行公平初始化。

格式：B=initcon(S, PR)

解析：当网络阈值学习函数为 learncon 时，需要采用该函数对阈值进行初始化，由于该函数返回的各神经元的阈值相等，因此称为公平初始化函数。该函数一般不用来对权值进行初始化。其参数说明如下。

S：神经元的个数；

PR：输入矢量取值范围的矩阵[Pmin Pmax]，默认值为[1 1]；

B：函数返回的初始阈值矢量。

【例 3-19】利用 initcon 函数对一个两神经元网络层的阈值进行初始化。

```
b=initcon(2)
```

```
b =
    5.4366
    5.4366
```

2．initzero

应用： 把权值和阈值初始化为零。

格式： W=initzero(S, PR)

　　　B=initzero(S, [1 1])

解析： 该函数把阈值和权值都初始化为零。其参数说明如下。

S：神经元个数；

PR：输入矢量取值范围的矩阵[Pmin Pmax]，采用 W=initzero(S, PR)式时，函数返回权值矩阵 W；采用 B=initzero(S, [1 1])式时，函数返回阈值矢量 B。

【例 3-20】利用 initzero 函数对一个具有两神经元二输入的网络层进行初始化。

```
W=initzero(2,[0 1;-1 1])
W =
     0     0
     0     0
B=initzero(2,[1 1])
B =
     0
     0
```

3．midpoint

应用： 把权值初始化为输入矢量值域的中心。

格式： W=midpoint(S, PR)

解析： 该函数把权值初始化为输入矢量值域的中心。其参数说明参见前面的介绍。

【例 3-21】利用 midpoint 函数对一个具有两神经元二输入的网络层权值进行初始化。

```
W=midpoint(2,[0 1;-1 1])
W =
    0.5000         0
    0.5000         0
```

4．randnr

应用： 对权值进行行归一化初始化。

格式： W=randnr(S, PR)

　　　W=randnr(S, P)

解析： 输入 S 为神经元个数；调用形式 W=randnr(S, PR)中的 PR 是表示输入矢量取值范围的矩阵，调用形式 W=randnr(S, P)中的 R 为输入矢量的维数。函数返回一个随机权值矩阵 W，矩阵中的每一行都经过归一化。

【例 3-22】产生一个 3 行 3 列的行归一化随机权值矩阵。

```
W=randnr(3,3)
W =
    0.9949   -0.0310   -0.0962
   -0.3976    0.5787   -0.7121
```

```
    0.2495    0.6120    0.7505
```

5．randnc

应用：对权值进行列归一化初始化。

格式：W=randnc(S, PR)

W=randnc(S, P)

解析：输入 S 为神经元个数；调用形式 W=randnc(S, PR)中的 PR 表示输入矢量取值范围的矩阵，调用形式 W=randnc(S, P)中的 R 为输入矢量的维数。函数返回一个随机权值矩阵 W，矩阵中的每一列都经过归一化。

【例 3-23】产生一个 3 行 3 列的列归一化随机权值矩阵。

```
W=randnc(3,3)
W =
   -0.1735    0.7240   -0.1545
    0.3621    0.4088    0.7137
    0.9158   -0.5556    0.6832
```

6．rands

应用：对权值和阈值进行随机初始化。

格式：W=rands(S, PR)

M=rands(S, R)

B=rands(S)

解析：调用形式 W=rands(S, PR)和 M=rands(S, R)都可以返回随机权值矩阵，调用形式 B=rands(S)返回阈值矢量。其他参数说明同上。

【例 3-24】产生一个 3 行 3 列的随机权值矩阵。

```
W=rands(3,3)
W =
   -0.1795   -0.2943   -0.7222
    0.7873    0.6263   -0.5945
   -0.8842   -0.9803   -0.6026
```

3.4 训练和自适应调整函数

网络的训练和自适应调整函数如表 3-4 所示。

表 3-4 网络的训练和自适应调整函数

函数名称	功　能
trainb	根据已设定的权值和阈值学习函数对网络进行批量训练
trainc	根据已设定的权值和阈值学习函数对网络进行循环训练
trainr	根据已设定的权值和阈值学习函数对网络进行随机训练
trains	根据已设定的权值和阈值学习函数对网络进行顺序训练
trainbr	采用贝叶斯正则化算法对网络进行训练
trainbfg	采用 BFGS 标准牛顿反向传播算法对网络进行训练
traincgb	采用 Powell-Beale 共轭梯度反向传播算法对网络进行训练

续表

函数名称	功　　能
traincgf	采用 Fletcher-Powell 共轭梯度反向传播算法对网络进行训练
traincgp	采用 Rolak-Ribiere 共轭梯度反向传播算法对网络进行训练
trainscg	采用尺度化共轭反向传播算法对网络进行训练
traingd	采用梯度下降反向传播算法对网络进行训练
traingda	采用自适应学习速率梯度下降反向传播算法对网络进行训练
traingdm	采用动量梯度下降反向传播算法对网络进行训练
traingdx	采用自适应学习速率动量梯度下降反向传播算法对网络进行训练
trainlm	采用 Levenberg-Marquardt 反向传播算法对网络进行训练
trainoss	采用一步正割反向传播算法对网络进行训练
trainrp	采用弹性反射传播算法对网络进行训练

1．trainb

应用：用于神经网络权值和阈值的训练。

格式：[NET, TR, Ac, E1]=trainb(net, Pd, T1, Ai, Q, TS, VV,TV)

info=trainb(code)

解析：trainb 函数实现对网络的批量式训练。所谓批量式训练，是指网络在接收全部样本矢量数据后，才对网络的权值和阈值进行调整。当训练函数属性 net.trainFcn 为'trainb'时，网络具有以下训练参数。

- net.trainParam.epochs：最大训练次数，网络每接收一轮输入数据并进行调整称为一次训练，该参数的默认值为 100；
- net.trainParam.goal：网络的性能目标，默认值为 0；
- net.trainParam.max_fail：最大验证失败次数，默认值为 5；
- net.trainParam.show：两次显示之间的训练次数，默认值为 25；
- net.trainParam.time：最长训练时间（以秒计），默认值为 inf。

其参数说明如下。

net：待训练的神经网络；

Pd：已延迟的输入信号；

T1：层目标；

Ai：初始的输入；

Q：批量；

TS：时间步长；

VV：确认向量或者为空矩阵；

TV：测试向量或者为空矩阵；

NET：函数返回值，训练后的神经网络；

TR：函数返回值，每一步的训练记录；

Ac：训练停止后，聚合层的输出；

E1：训练停止后的误差。

函数的调用形式 info=trainb(code)将返回训练函数的有关信息，code 字符串的取值为：

- 'pnames'：返回训练参数的名称；
- 'pdefaults'：返回训练参数的默认值。

网络属性设置：

newlin 函数建立的网络采用 trainb 作为训练函数，如果要使自定义网络利用 trainb 函数进行训练，可作如下设置：

（1）将网络训练函数 net.trainFcn 设置为'trainb'；

（2）设置网络训练函数的参数 net.trainParam；

（3）设置各输入权值、网络权值和阈值的学习函数 net.inputWeights{i,j}.learnFcn、net.layerWeights{i,j}.learnFcn 和 net.biases{i}.learnFcn；

（4）设置各权值和阈值学习函数的参数 net.inputWeights{i,j}.learnParam、net.layerWeights{i,j}.learnParam 和 net.biases{i}.learnParam。

2：trainc

应用：根据已设定的权值和阈值学习函数对网络进行循环训练。

格式：[net, TR, Ac, E1]=trainc(net, Pd, T1, Ai, Q, TS, VV, TV)

info=trainc(code)

解析：trainc 函数对网络进行循环训练。所谓循环训练，是指网络每接收一个输入数据，就要对权值和阈值进行一次调整，而且输入数据是以循环队列的形式进行排列的。网络训练函数属性 net.trainFcn 为'trainc'时，网络的训练参数和 trainb 参数含义相同。其输入量和返回量的含义同前面相同参数的含义。

网络属性设置：

newp 函数建立的网络采用 trainc 作为训练函数，如果要使自定义网络利用 trainc 函数进行训练，可作如下设置：除了将网络训练函数 net.trainFcn 设置为'trainc'外，其他设置与 trainb 一样。

3. trainr

应用：根据已设定的权值和阈值学习函数对网络进行随机训练。

格式：[net, TR, Ac, E]=trainr(net, Pd, T1, Ai, Q, TS, VV, TV)

info=trainr(code)

解析：train 函数对网络进行随机训练。所谓随机训练，是指网络每接收一个输入数据，就要对权值和阈值进行一次调整，而且输入数据是以随机顺序排列的。当网络训练函数的属性 net.trainFcn 为'trainr'时，网络的训练参数与 trainc 函数相同。

其参数含义也与 trainc 相同。

网络属性设置：

newc 和 newsom 函数建立的网络都采用 trainr 作为训练函数，如果要使自定义网络利用 trainr 函数进行训练，可作如下设置：除了将网络训练函数 net.trainFcn 设置为'trainr'外，其他的设置与 trainc 相同。

4. trains

应用：根据已设定的权值和阈值学习函数对网络进行顺序训练。

格式：[net, TR, Ac, E1]=trains(net, Pd, T1, Ai, Q, TS, VV, TV)

info=trainr(code)

解析：trains 函数依据已设定的权值和阈值学习函数对网络进行顺序训练。所谓顺序训练，是指输入数据按排列顺序依次进入网络，网络每接收一个输入，就对权值和阈值进行一次调整。

该函数多设置为网络的自适应调整函数。当网络的自适应调整函数属性 net.adaptFcn 为'trains'时，网络的自适应调整参数如下。

net.adaptParam.passes：自适应调整次数。网络接收一轮输入数据后并对权值和阈值进行更新称为一次调整，该参数默认值为 1。

当网络训练函数属性 net.trainFcn 为'trains'时，网络的训练参数如下。

net.trainParam.passes：训练次数，默认值为 1。

trains 函数在调用时各输入量和返回量的含义参见 trainb 函数。函数输出的训练记录 TR 由网络训练的时间步数 TR.timesteps 和训练性能 TR.perf 两项组成。

网络属性设置：

newp 和 newlin 函数建立的网络都采用 trains 函数作为自适应调整函数，如果要使自定义网络利用 trains 函数进行调整，可作如下设置：除了将网络训练函数 net.adaptFcn 设置为'trains'以及设置网络训练函数的参数 net.adaptParam 外，其他设置和 trainb 一样。

5．trainbr

应用：采用贝叶斯正则化算法对网络进行训练。

格式：[net, TR, Ac, E1]=trainbr(net, Pd, T1, Ai, Q, TS, VV, TV)

info=trainr(code)

解析：trainbr 函数依据 Levenberg-Marquardt 优化理论对网络的权值和阈值进行调整。所谓 Bayesian 正则化，是指为了提高网络的推广能力，训练过程中要建立一个由各层输出误差、权值和阈值构成的特殊性能参数（该参数的定义类似于规则化均方误差函数 msereg），通过调整权值和阈值，使该参数最小化。trainbr 函数的优点在于可以提高网络推广能力，而且不会出现“过度训练”的情况。当网络输入和目标矢量的取值为[-1 1]时，trainbr 函数可以达到最好的工作效果，因此在利用 trainbr 函数对网络进行训练之前通常应预先对样本数据作归一化处理。

当网络的训练函数 net.trainFcn 为'trainbr'时，网络的训练参数为：

- net.trainParam.epochs：训练次数，默认值为 100；
- net.trainParam.goal：网络性能目标，默认值为 0；
- net.trainParam.mu：Marquardt 调整参数，默认值为 0.005；
- net.trainParam.mu_dec：mu 的下降因子，默认值为 0.1；
- net.trainParam.mu_inc：mu 的上升因子，默认值为 10；
- net.trainParam.mu_max：mu 的最大值，默认值为 1e-10；
- net.trainParam.max_fail：最大验证失效次数，默认值为 5；
- net.trainParam.mem_reduc：该参数用于权衡计算雅可比矩阵时占用的内存空间和计算速度，默认值为 1。当该因子为 1 时，计算时占用的内存最大，但计算速度最快，当该因子增大时，计算时占用的内存减小，但计算的速度也随之减慢；
- net.trainParam.min_grad：性能函数的最小梯度，默认值为 1e-10；
- net.trainParam..show：两次显示之间的训练次数，默认值为 25；
- net.trainParam.time：最长训练时间（以秒计），默认值为 inf。

trainbr 函数在调用时各输入量和返回量的含义参见 trainb 函数。

网络属性设置：

newff、newcf 和 newelm 函数建立的网络都可以采用 trainbr 函数作为训练函数。只要网络的加权函数、输入函数和传递函数是可微分的，那么该网络就可以利用 trainbr 函数进行训练，

如果要使网络利用 trainbr 函数进行训练，可作如下设置：

（1）将网络训练函数 net.trainFcn 设置为'trainbr'；

（2）设置网络训练函数的参数 net.trainParam。

6. trainbfg

应用： 采用 BFGS 标准牛顿反向传播算法对网络进行训练。

格式： [net, TR, Ac, El]=trainbfg(net, Pd, Tl, Ai, Q, Ts, VV, TV)

info=trainbfg(code)

解析： trainbfg 函数依据 BFGS 标准牛顿反向传播算法对网络的权值和阈值进行调整，该算法收敛较快，但占用的内存比共轭梯度算法要大，因此适用于较小规模的网络。

trainbfg 函数及其他一些采用标准牛顿算法和共轭梯度算法的训练函数都要使用线搜索函数，因此当网络的训练函数 net.trainFcn 设为此类函数时，网络的训练参数将由常规参数和线搜索参数组成，其中常规参数包括：

- net.trainParam.epochs：训练次数，默认值为 100；
- net.trainParam.goal：网络性能目标，默认值为 0；
- net.trainParam.max_fail：最大验证失败次数，默认值为 5；
- net.trainParam.min_grad：性能函数的最小梯度，默认值为 1e-6；
- net.trainParam.show：两次显示之间的训练次数，默认值为 25；
- net.trainParam.time：最长训练时间（以秒计），默认值为 inf。

线搜索参数包括：

- net.trainParam.searchFcn：训练时采用的线搜索方法，默认值为'srchcha'；
- net.trainParam.scal_tol：确定线搜索误差的尺度因子，默认值为 20，初始步长 delta 除以该参数后得到线搜索结果的误差；
- net.trainParam.alpha：确定性能函数下降是否足够大的尺度因子，默认值为 0.001；
- net.trainParam.beta：确定步长是否足够大的尺度因子，默认值为 0.1；
- net.trainParam.delta：区间定位时的初始步长，默认值为 0.01；
- net.trainParam.gama：避免性能函数下降过小的参数，默认值为 0.1；
- net.trainParam.low_lim：步长变化下界，默认值为 0.1；
- net.trainParam.up_lim：步长变化上界，默认值为 0.5；
- net.trainParam.maxstep：最大步长，默认值为 100；
- net.trainParam.minstep：最小步长，默认值为 1.0e-6；
- net.trainParam.bmax：最大步长（与 maxstep 在不同搜索算法中分别使用），默认值为 26。

网络性能设置：

newff、newcf 和 newelm 函数建立的网络都可以采用 trainbfg 函数作为训练函数。只要网络的加权函数、输入函数和传递函数是可微分的，那么该网络就可以利用 trainbfg 函数进行训练，如果要使网络利用该函数进行训练，可作如下设置：

（1）将网络训练函数 net.trainFcn 设置为'trainbfg'；

（2）设置网络训练函数的参数 net.trainParam。

7. traingd

traingd 函数使用梯度下降算法训练神经网络，通过 net.trainParam 可以查看 traingd 函数网络训练的相关参数。

```
P=[1.24 1.36 1.38 1.38 1.38 1.4 1.48 1.54 1.56 1.14 1.18 1.2 1.26 1.28 1.3;
   1.72 1.74 1.64 1.82 1.9 1.7 1.82 1.82 2.08 1.78 1.96 1.86 2.0 2.0 1.96];
net=newff(minmax(P),[5,1],{'logsig','purelin'},'traingd');
net.trainParam
ans =
        epochs: 100
          goal: 0
            lr: 0.0100
      max_fail: 5
      min_grad: 1.0000e-010
          show: 25
          time: Inf
```

使用 traingd 函数进行蠓虫分类，在 MATLAB 命令窗口中输入

```
net=newff(minmax(P),[5,1],{'logsig','purelin'},'traingd');
net.trainParam.show=50;
net.trainParam.lr=0.2;
net.trainParam.epochs=300;
net.trainParam.goal=0.01;
[net,tr]=train(net,P,T);
%回代检验
A=sim(net,P);
%测试样本检验
a=sim(net,P);
```

检验结果如下

```
a =
    0.6782      0.5867      0.4327
```

网络训练过程均方差下降曲线如图 3-7 所示。为了比较各种不同训练函数的特点，在后面的实例中，将统一使用蠓虫分类问题的训练样本数据和测试样本数据。

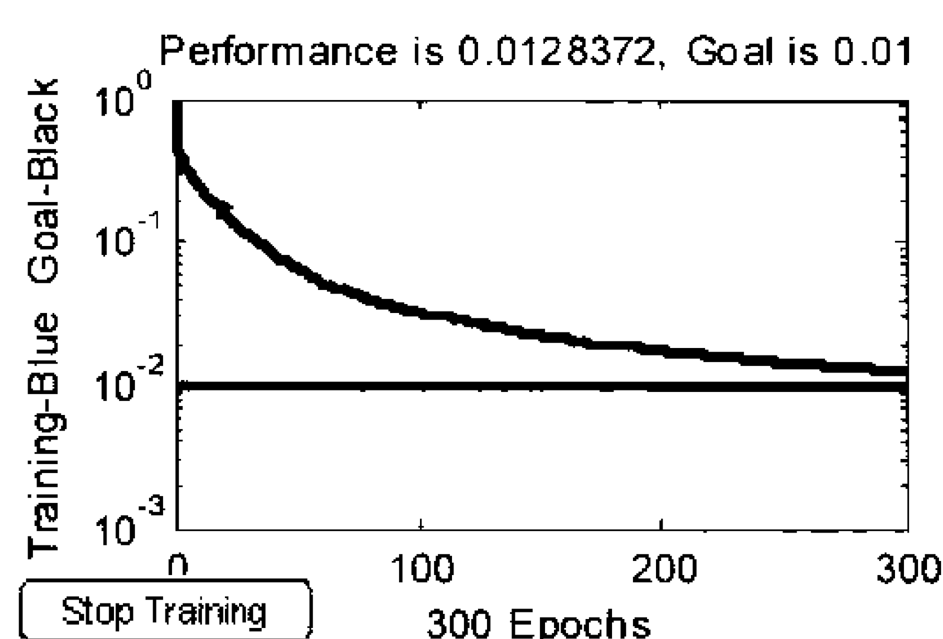


图 3-7　网络训练过程均方差(mse)下降曲线

8．traingdm

同 traingd 函数相似，traingdm 采用动量梯度下降法训练 BP 网络，通过设置 net.trainParam.mc 来设置动量因子的大小。

```
net=newff(minmax(P),[5,1],{'logsig','purelin'},'traingdm');
net.trainParam
ans =
        epochs: 100
          goal: 0
```

```
            lr: 0.0100
    max_fail: 5
            mc: 0.9000
    min_grad: 1.0000e-010
         show: 25
         time: Inf
```

在 MATLAB 命令行窗口中输入

```
net=newff(minmax(P),[5,1],{'logsig','purelin'},'traingdm');
net.trainParam.show=50;
net.trainParam.lr=0.1;
net.trainParam.mc=0.9;
net.trainParam.epochs=300;
net.trainParam.goal=0.01;
[net,tr]=train(net,P,T);
%回代检验
A=sim(net,P);
%测试样本检验
a=sim(net,P);
```

如图 3-8 所示为 traingdm 函数训练过程曲线。训练后的网络对测试样本的输出为

```
a =
    0.6705       0.5641       0.4307
```

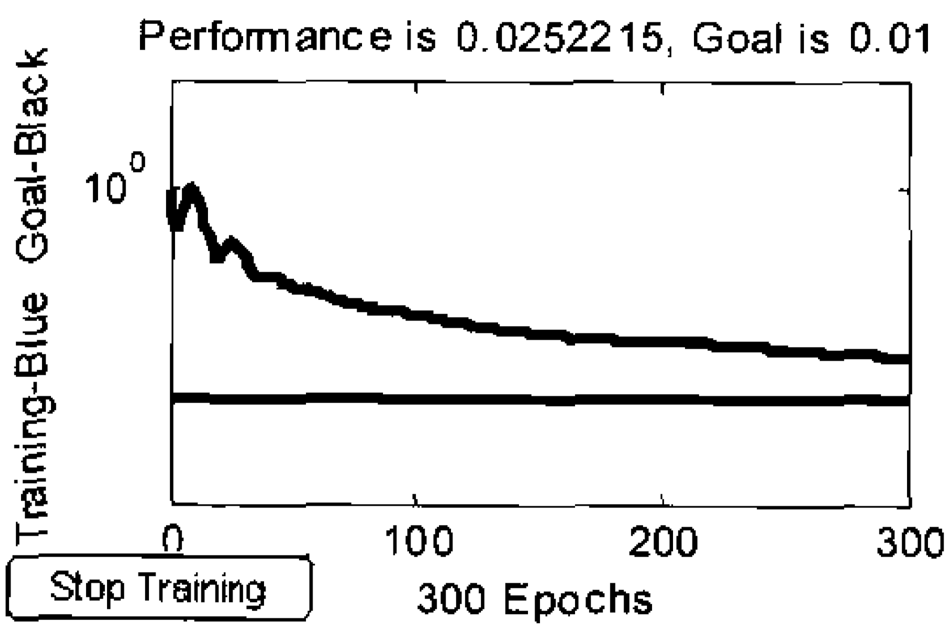


图 3-8 traingdm 函数训练过程曲线

9. traingda

当 BP 网络使用 traingd 函数和 traingdm 函数训练时，学习速率在训练过程中保持恒定不变，那么训练结果对学习速率的灵敏度影响较大，不同的学习速率对网络的训练结果影响较大。如果学习速率过大，那么网络将变得不稳定；如果学习速率过小，那么网络收敛速度慢，训练时间大大加长。

因此，对于给定训练样本和目标样本，必须首先确定最优的学习速率。而 traingda 函数为变学习速率的网络训练算法，可以有效地克服学习速率难以确定的缺点，在训练过程中自适应改变学习速率的大小。

```
net=newff(minmax(P),[5,1],{'logsig','purelin'},'traingda');
net.trainParam
ans =
         epochs: 100
            goal: 0
               lr: 0.0100
```

```
          lr_inc: 1.0500
          lr_dec: 0.7000
        max_fail: 5
    max_perf_inc: 1.0400
        min_grad: 1.0000e-006
            show: 25
            time: Inf
```

在 MATLAB 命令窗口中输入以下程序段

```
net=newff(minmax(P),[5,1],{'logsig','purelin'},'traingda');
net.trainParam.show=50;
net.trainParam.lr=0.1;
net.trainParam.lr_inc=1.05;
net.trainParam.lr_dec=0.85;
net.trainParam.epochs=300;
net.trainParam.goal=0.01;
[net,tr]=train(net,P,T);
%回代检验
A=sim(net,P);
%测试样本检验
a=sim(net,P);
```

如图 3-9 所示为 traingda 函数训练过程曲线，BP 网络对测试样本的输出结果为

```
a =
    0.4192      0.5941      0.7764
```

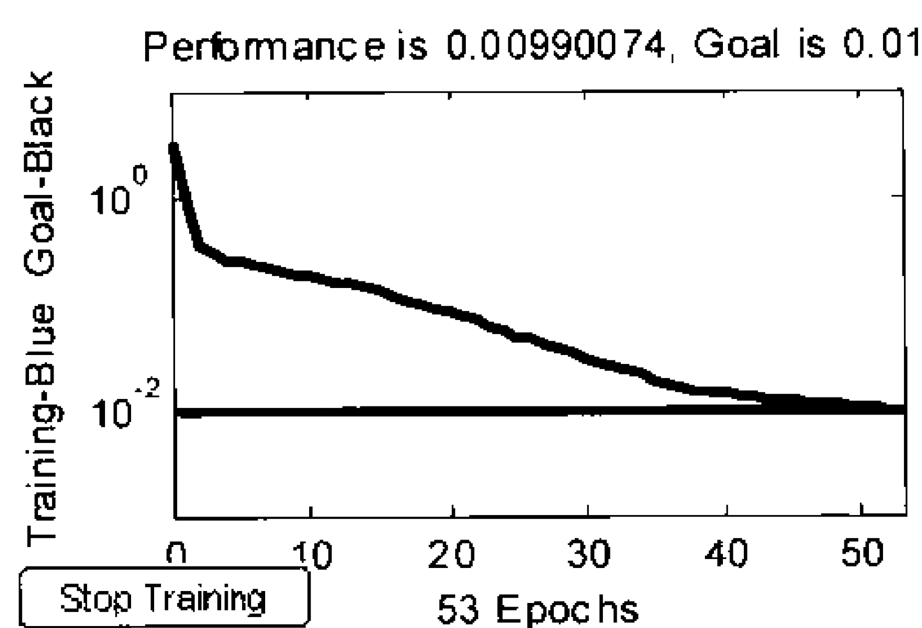


图 3-9　traingda 函数训练过程曲线

10. traingdx

traingdx 函数结合了自适应改变学习速率和动量法，和 traingda 函数完全相同，只是增加了一个动量因子参数 mc。

```
net=newff(minmax(P),[5,1],{'logsig','purelin'},'traingdx');
net.trainParam
ans =
          epochs: 100
            goal: 0
              lr: 0.0100
          lr_dec: 0.7000
          lr_inc: 1.0500
        max_fail: 5
    max_perf_inc: 1.0400
```

```
          mc: 0.9000
    min_grad: 1.0000e-006
        show: 25
        time: Inf
```

在 MATLAB 命令行窗口中输入

```
net=newff(minmax(P),[5,1],{'logsig','purelin'},'traingdx');
net.trainParam.show=50;
net.trainParam.lr=0.1;
net.trainParam.lr_inc=1.05;
net.trainParam.lr_dec=0.85;
net.trainParam.mc=0.9;
net.trainParam.epochs=300;
net.trainParam.goal=0.01;
[net,tr]=train(net,P,T);
%回代检验
A=sim(net,P);
%测试样本检验
a=sim(net,P);
```

如图 3-10 所示为 traingdx 函数训练过程曲线，测试样本的输出结果为

```
a =
    0.5880      0.6223      0.5236
```

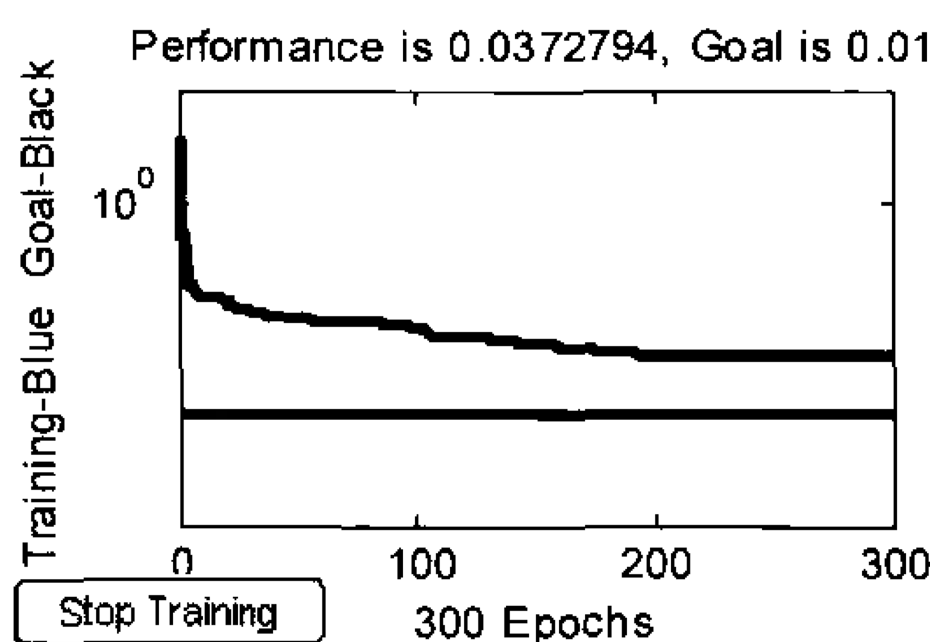


图 3-10 traingdx 函数训练过程曲线

11．trainrp

前面介绍的所有训练函数大都采用梯度的变化量作为权重和阈值的变化依据，但是当函数在极值旁非常平坦时，此时梯度的变化量非常小，那么权重和阈值的变化将非常小，过小的梯度变化一方面导致网络训练收敛速度非常慢，另一方面又使网络无法达到训练的性能要求。

trainrp 函数可以克服这方面的缺点，trainrp 函数使用误差梯度的方向来确定权重和阈值的变化，而误差梯度的大小对权重和阈值的变化没有任何作用，权重和阈值的变化量由另外两个参数 delt_inc 和 delt_dec 来确定。trainrp 函数的训练参数设置为

```
net=newff(minmax(P),[5,1],{'logsig','purelin'},'trainrp');
net.trainParam
ans =
      epochs: 100
        show: 25
        goal: 0
        time: Inf
    min_grad: 1.0000e-006
```

```
    max_fail: 5
    delt_inc: 1.2000
    delt_dec: 0.5000
      delta0: 0.0700
    deltamax: 50
```

在 MATLAB 命令行窗口中输入

```
net=newff(minmax(P),[5,1],{'logsig','purelin'},'trainrp');
net.trainParam.show=50;
net.trainParam.epochs=300;
net.trainParam.goal=0.01;
net.trainParam.delt_inc=1.5;
net.trainParam.delt_dec=0.8;
[net,tr]=train(net,P,T);
%回代检验
A=sim(net,P);
%测试样本检验
a=sim(net,P);
```

如图 3-11 所示为 trainrp 函数训练过程曲线，测试样本输出结果为

```
a =
    0.7422      0.8566      0.2832
```

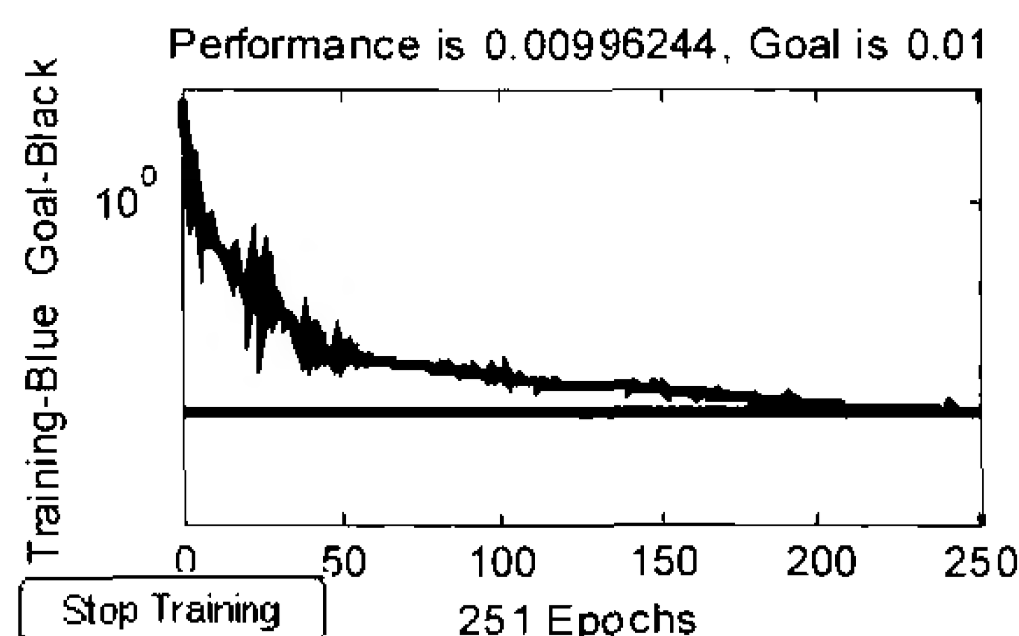


图 3-11　trainrp 函数训练过程曲线

12. traincgf

前面介绍的所有 BP 网络训练函数大都采用负梯度方向，即按误差下降方向进行搜索，这样的搜索策略可以使目标函数即网络训练误差下降速率最快，但是却不能保证网络最快收敛，因此提出了共轭梯度算法，搜索方向为共轭梯度方向。

共轭梯度算法训练函数包括 traincgf 函数、traincgp 函数、traincgb 函数和 trainscg 函数。接下来将介绍共轭梯度算法函数。

```
net=newff(minmax(P),[5,1],{'logsig','purelin'},'traincgf');
net.trainParam
ans =
       epochs: 100
         show: 25
         goal: 0
         time: Inf
     min_grad: 1.0000e-006
     max_fail: 5
    searchFcn: 'srchcha'
```

```
scale_tol: 20
      alpha: 1.0000e-003
       beta: 0.1000
      delta: 0.0100
       gama: 0.1000
    low_lim: 0.1000
     up_lim: 0.5000
    maxstep: 100
    minstep: 1.0000e-006
       bmax: 26
```

在 MATLAB 命令行窗口中输入

```
net=newff(minmax(P),[5,1],{'logsig','purelin'},'traincgf');
net.trainParam.show=50;
net.trainParam.epochs=300;
net.trainParam.goal=0.01;
net.trainParam.searchFcn='srchbre';
[net,tr]=train(net,P,T);
%回代检验
A=sim(net,P);
%测试样本检验
a=sim(net,P);
```

traincgf 函数 BP 网络训练过程曲线如图 3-12 所示。网络训练迭代过程信息及测试样本的输出结果如下

```
a =
    0.4435      0.5370      0.5611
```

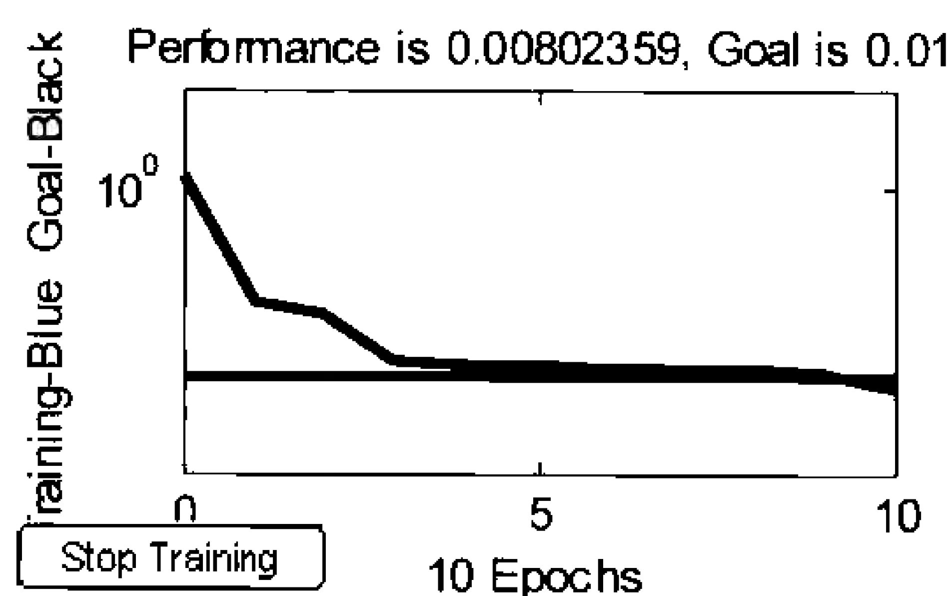


图 3-12 traincgf 函数 BP 网络训练过程曲线

13. traincgp

与 traincgf 函数相同，traincgp 函数采用共轭梯度算法进行 BP 网络训练，traincgp 函数的训练参数与 traincgf 完全相同，具体参数设置见 traincgf 函数。

使用 traincgp 函数重新训练网络，在 MATLAB 命令行窗口中输入

```
net=newff(minmax(P),[5,1],{'logsig','purelin'},'traincgp');
net.trainParam.show=50;
net.trainParam.epochs=300;
net.trainParam.goal=0.01;
net.trainParam.searchFcn='srchhyb';
[net,tr]=train(net,P,T);
%回代检验
```

```
A=sim(net,P);
%测试样本检验
a=sim(net,P);
```

traincgp 函数 BP 网络训练过程曲线如图 3-13 所示。网络训练迭代过程信息及测试样本的输出结果如下。

```
a =
    0.4735      0.5419      0.5998
```

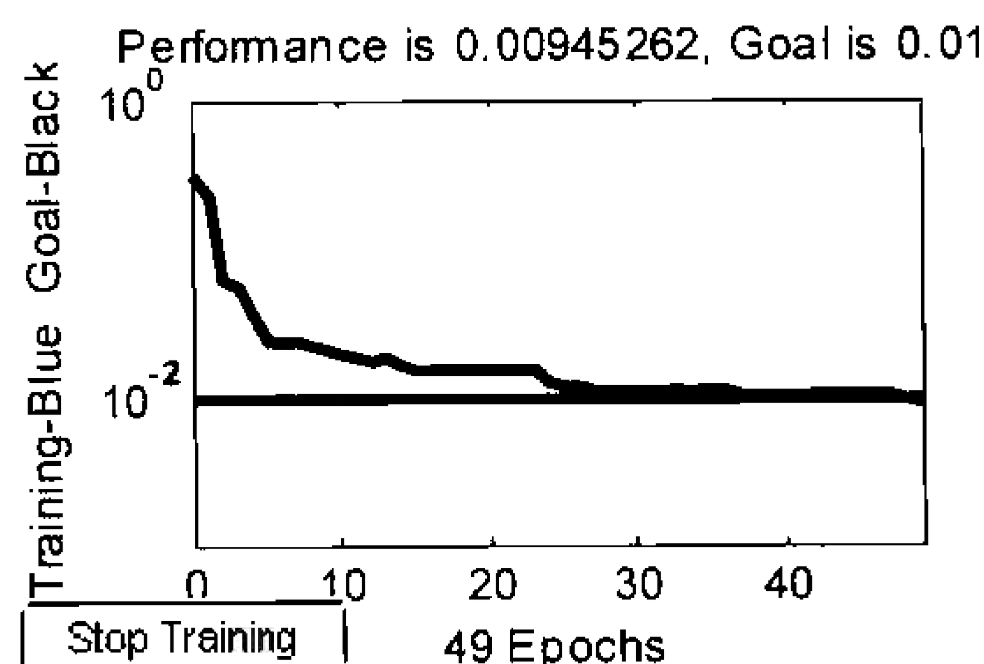


图 3-13　traincgp 函数 BP 网络训练过程曲线

读者可以参照 traincgf 和 traincgp 函数，使用共轭梯度算法训练函数 traincgf 和 traincgp 进行 BP 网络训练，体会函数间的差异。

14．traincgb

与 traincgf 函数相同，traincgb 函数采用共轭梯度算法进行 BP 网络训练，traincgb 函数的训练参数与 traincgf 完全相同，具体参数设置见 traincgf 函数。

```
net=newff(minmax(P),[5,1],{'logsig','purelin'},'traincgb');
net.trainParam.show=50;
net.trainParam.epochs=300;
net.trainParam.goal=0.01;
net.trainParam.searchFcn='srchcha';
[net,tr]=train(net,P,T);
%回代检验
A=sim(net,P);
%测试样本检验
a=sim(net,P);
```

traincgb 函数 BP 网络训练过程曲线如图 3-14 所示。网络训练迭代过程信息及测试样本的输出结果如下。

```
a =
    0.4935      0.6419      0.4998
```

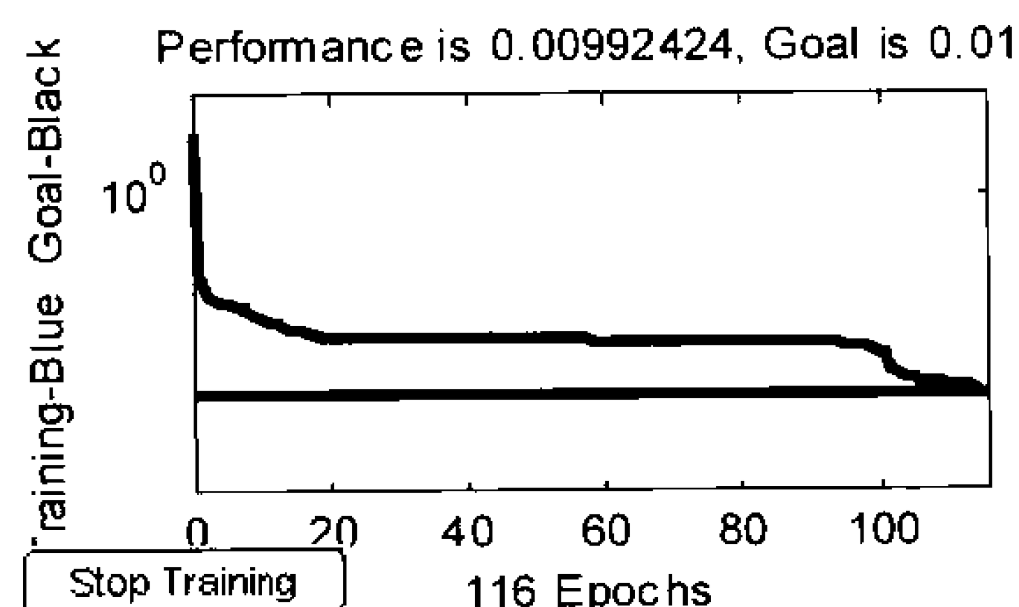


图 3-14　traincgb 函数 BP 网络训练过程曲线

15. trainlm

trainlm 函数同拟牛顿法相似，使用一阶 Jacobian 矩阵近似计算二阶 Hessian 矩阵，避免了网络训练误差的二阶导数。通过以下命令获取 trainlm 函数的参数设置

```
net=newff(minmax(P),[5,1],{'logsig','purelin'},'trainlm');
net.trainParam
trainlm 函数的训练参数如下：
ans =
         epochs: 100
           goal: 0
       max_fail: 5
      mem_reduc: 1
       min_grad: 1.0000e-010
             mu: 1.0000e-003
         mu_dec: 0.1000
         mu_inc: 10
         mu_max: 1.0000e+010
           show: 25
           time: Inf
```

在 MATLAB 命令行窗口中输入

```
net=newff(minmax(P),[5,1],{'logsig','purelin'},'trainlm');
net.trainParam.show=50;
net.trainParam.epochs=300;
net.trainParam.goal=0.01;
net.trainParam.mu_dec=0.1;
net.trainParam.mu_inc=7;
[net,tr]=train(net,P,T);
%回代检验
A=sim(net,P);
%测试样本检验
a=sim(net,P);
```

trainlm 函数 BP 网络训练过程曲线如图 3-15 所示。网络训练迭代过程信息及测试样本的输出结果如下。

```
a =
    0.7015      0.6898      0.2336
```

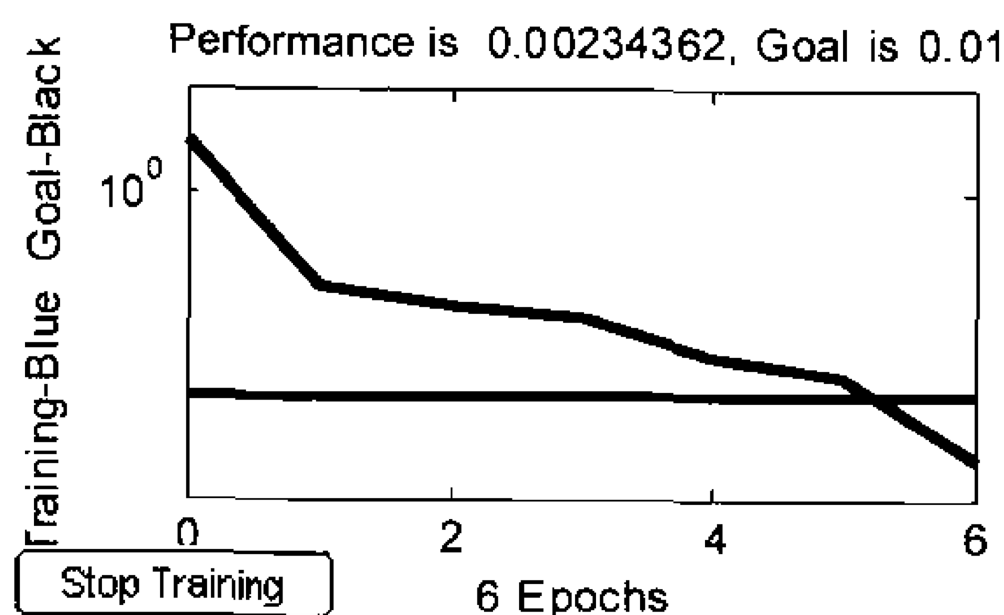


图 3-15 trainlm 函数 BP 网络训练过程曲线

16. trainscg

trainscg 函数是共轭梯度算法的一种变形，该算法结合了 Levenberg-Marquardt 算法中的模

型置信区间方法和共轭梯度算法，避免了耗时巨大的线搜索过程，从而提高了网络的训练速度。通过以下命令获取 trainscg 函数的参数设置。

```
net=newff(minmax(P),[5,1],{'logsig','purelin'},'trainscg');
net.trainParam
ans =
        epochs: 100
          show: 25
          goal: 0
          time: Inf
      min_grad: 1.0000e-006
      max_fail: 5
         sigma: 5.0000e-005
        lambda: 5.0000e-007
```

17．trainoss

trainoss 函数依据一步正割反向传播算法对网络的权值和阈值进行调整，该函数是一种标准牛顿算法。trainoss 函数占用的内存比共轭梯度算法大，但却小于 BFGS 算法（参见 trainbfg 函数），因此可以认为是两者的折中。通过以下命令获取 trainscg 函数的参数设置。

```
net=newff(minmax(P),[5,1],{'logsig','purelin'},'trainoss');
net.trainParam
ans =
        epochs: 100
          show: 25
          goal: 0
          time: Inf
      min_grad: 1.0000e-006
      max_fail: 5
     searchFcn: 'srchbac'
     scale_tol: 20
         alpha: 1.0000e-003
          beta: 0.1000
         delta: 0.0100
          gama: 0.1000
       low_lim: 0.1000
        up_lim: 0.5000
       maxstep: 100
       minstep: 1.0000e-006
          bmax: 26
```

在 MATLAB 命令行窗口中输入

```
net=newff(minmax(P),[5,1],{'logsig','purelin'},'trainoss');
net.trainParam.show=50;
net.trainParam.epochs=300;
net.trainParam.goal=0.01;
net.trainParam.searchFcn='srchbac';
[net,tr]=train(net,P,T);
%回代检验
```

```
A=sim(net,P);
%测试样本检验
a=sim(net,P);
```

trainoss 函数 BP 网络训练过程曲线如图 3-16 所示。网络训练迭代过程信息及测试样本的输出结果如下。

```
a =
      0.5435      0.4680      0.7021
```

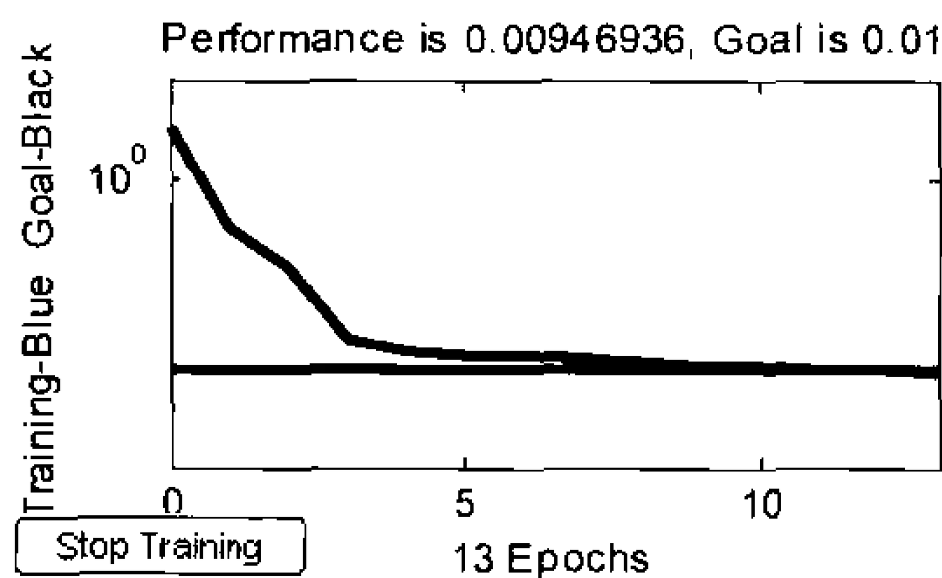


图 3-16 trainoss 函数 BP 网络训练过程曲线

3.5 神经网络的学习函数

网络权值和阈值的学习函数如表 3-5 所示。这些函数在程序中不被直接调用，当网络利用函数 train、trainc、trainr 和 trains 进行训练时，这些训练函数将根据已经设置好的权值和阈值学习函数的属性调用相应的学习函数。

表 3-5 学习函数

函数名称	功 能	函数名称	功 能
learncon	公平阈值学习函数	learnlv1	LVQ1 权值学习函数
learngd	梯度下降权值和阈值学习函数	learnlv2	LVQ2 权值学习函数
learngdm	动量梯度下降权值和阈值学习函数	learnos	星外权值学习函数
learnh	Hebb 权值学习函数	learnp	感知器权值和阈值学习函数
learnhd	退化 Hebb 权值学习函数	learnpn	归一化感知器权值和阈值学习函数
learnis	星内权值学习函数	learnsom	自组织映射网络权值学习函数
learnk	Kohonen 权值学习函数	learnwh	Widrow-Hoff 权值和阈值学习函数

1. learncon

应用： 公平阈值学习函数。

格式： [dB, LS]=learncon(B, P, Z, N, A, T, E, gW, gA, D, LP, LS)

info=learncon(code)

解析： learncon 函数是一种阈值学习函数，常用于调整竞争层网络中神经元的阈值，该函数通过建立一个公正参数，以保证小输出神经元的阈值大幅度增加，大输出神经元的阈值小幅度增加。该函数中阈值的调整量仅由神经元的原始阈值和输出决定。

在函数调用形式[dB, LS]=learncon(B, P, Z, N, A, T, E, gW, gA, D, LP, LS)中，B 为网络某层的阈值矢量；P 为全 1 矩阵；Z 为经过加权函数变换后的加权输入矢量；N 为加权输入经过输入函数变换后的神经元传递函数的阈值的梯度矢量；A 为该层网络的输出矢量；T 为目标矢量；

E 为误差矢量；gW 为网络性能对于阈值的梯度矢量；gA 为网络性能对于该层输出的梯度矢量；D 为神经元的距离矩阵；LP 为学习参数，learncon 函数的学习参数是由学习速率 LP.lr 构成的，默认值为 0.001；LS 为学习状态，初始状态时取值为[ ]。函数返回阈值调整量 dB 和当前学习状态 LS。

函数的调用形式 info=learncon(code)返回该学习函数的有关信息。code 字符串的取值如下。

- 'pnames'：返回学习参数的名称；
- 'pdefaults'：返回学习参数的默认值；
- 'needg'：返回学习函数是否要利用梯度矢量 gW 或 gA，当用到梯度矢量时，函数返回 1。

【例 3-25】假定 ***A*** 和 ***B*** 分别为两神经元网络层的输出矢量和阈值矢量。

```
A=[0.8956;0.2341];
B=[0.7432;0.4589];
```

设置好学习速率并利用公平学习规则对阈值进行更新。

```
LP.lr=0.6;
dB=learncon(B,[ ],[ ],[ ],A,[ ],[ ],[ ],[ ],[ ],LP,[ ])
```

阈值更新量为

```
dB =
     0.6157
     0.6241
```

2. learngd

应用： 梯度下降权值和阈值学习函数。

格式： [dW, LS]=learngd(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

[dB, LS]=learngd(B, P, Z, N, A, T, E, gW, gA, D, LP, LS)

info=learngd(code)

解析： learngd 函数采用梯度下降方法对权值和阈值进行调整，即权值和阈值的调整量 dW 为学习速率 lr 和梯度 gW 的乘积

$$dW(i, j)=lr*gW(i, j)$$

在函数调用形式[dW, LS]=learngd(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)中，W 为权值矩阵；P 为层输入矢量；Z 为层输入经过加权函数变换后的加权输入矢量；N 为加权输入经过输入函数计算后得到的神经元传递函数的输入矢量；A 为该层神经元输出矢量；T 为目标矢量；E 为误差矢量；gW 为网络性能对于权值的梯度矢量；gA 为网络性能对于该层的输出梯度矢量；D 为神经元的距离矩阵；LP 为学习参数，learngd 函数的学习参数是由学习速率 LP.lr 构成的，默认值为 0.01；LS 为学习状态，初始值为[]。函数返回阈值调整量 dW 和当前学习状态 LS。函数的其余两种参数调用形式见函数 learncon。

【例 3-26】对一个 3 输入的两神经元网络层随机产生梯度矢量 gW。

```
gW=rand(2,3)
gW =
     0.9501     0.6068     0.8913
     0.2311     0.4860     0.7621
```

设置好学习速率并利用梯度下降法对权值进行更新。

```
LP.lr=0.6;
```

```
dW=learngd([ ],[ ],[ ],[ ],[ ],[ ],[ ],gW,[ ],[ ],LP,[ ])
dW =
      0.5701      0.3641      0.5348
      0.1387      0.2916      0.4573
```

3. learngdm

应用：动量梯度下降权值和阈值学习函数。

格式：[dW, LS]=learngd(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

[dB, LS]=learngd(B, P, Z, N, A, T, E, gW, gA, D, LP, LS)

info=learngd(code)

解析：learngdm 函数采用动量梯度下降法对权值和阈值进行调整，即权值和阈值的调整值 dW 由动量因子 mc、前一次学习时的调整量 dWprev、学习速率 lr 和梯度 gW 共同确定。

$$dW(i, j)=mc*dWprev(i, j) * (1-mc) *lr*gW(i, j)$$

函数调用时各输入和返回量的含义参见函数 learncon 和 learngd，其中前一次学习时的权值及阈值调整量 dWprev 存储在学习状态参数 LS.dW 中。learngdm 函数的学习参数包括如下。

- LP.lr：学习速率，默认值为 0.01；
- LP.mc：动量常数，默认值为 0.9。

【例 3-27】设定权值梯度矩阵 gW 和上一次的权值调整量 LS.dw 为

```
gW=[0.6782 0.9637 0.7145 0.1287 0.7963 0.4056];
LS.dw=[0.8963 0.1287 0.1982 0.0089 0.2365 0.7421];
```

设置学习参数并利用动量梯度下降法对权值进行更新。

```
LP.lr=0.6;
LP.mc=0.8;
dW=learngdm([],[],[],[],[],[],[],gW,[],[],LP,LS)
dW =
      0.7984      0.2186      0.2443      0.0226      0.2848      0.6424
```

4. learnh

应用：Hebb 权值学习函数。

格式：[dW, LS]=learnh(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

info=learnh(code)

解析：learnh 函数采用 Hebb 联想学习规则对神经元权值进行调整，即两个神经元的网络权值正比于它们的活动值，所以当两个神经元的输出同时为高时，这两个神经元之间将具有较大的网络权值，权值调整值 dW 由学习速率 lr、神经元输入 P 和输出 A 共同确定。

$$dW(i, j)=lr*A(i) *P(j)$$

函数调整时各输入量和返回值的含义参见函数 learngd。该函数的学习参数是由学习速率 LP.lr 构成的，默认值为 0.01。

【例 3-28】假定 ***P*** 和 ***A*** 分别为一个两神经元 3 输入网络层的输入和输出矢量。

```
P=[0.3578;0.1986;0.0135];
A=[0.7852;0.4423];
```

设置学习速率并利用 Hebb 学习规则更新权值。

```
LP.lr=0.6;
```

```
dW=learnh([],P,[],[],A,[],[],[],[],[],LP,[])
dW =
     0.1686     0.0936     0.0064
     0.0950     0.0527     0.0036
```

5．learnhd

应用：退化 Hebb 权值学习函数。

格式：[dW, LS]=learnhd(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

info=learnhd(code)

解析：learnhd 函数是 Hebb 联想学习规则的变形，该函数中通过增加一个附加项来限制权值的规模，该附加项由衰减率 dr 和当前权值 W 确定，这时权值调整量为

$$dW(i, j)=lr*A(i)*P(j)-dr*W(i, j)$$

其中，lr 为学习速率，P 为神经元输入，A 为输出。

函数调用时各输入量和返回量的含义参见函数 learngd。该函数的学习参数包括如下。

- LP.lr：学习速率，默认值为 0.1；
- LP.dr：衰减率，默认值为 0.01。

【例 3-29】根据下列输入矢量 ***P***、输出矢量 ***A*** 和原始权值矩阵 ***W***，利用 learnhd 函数对神经元权值进行学习。

```
P=[0.3578;0.1986;0.0135];
A=[0.7852;0.4423];
W=[0.6782 0.0198 0.3753;0.8217 0.6875 0.8631];
```

设置学习速率、衰减率并更新权值。

```
LP.lr=0.6;
LP.dr=0.05;
dW=learnhd(W,P,[],[],A,[],[],[],[],[],LP,[])
dW =
     0.1347     0.0926     -0.0124
     0.0539     0.0183     -0.0396
```

6．learnis

应用：星内权值学习函数。

格式：[dW, LS]=learnis(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

info=learnis(code)

解析：learnis 函数利用星内联想式学习方法对权值进行调整。在该方法中，当神经元具有高输出时，权值将向当前输入矢量进行逼近调整，这样当网络再输入同一矢量时，该神经元将出现明显的高输出。learnis 函数中的权值调整原理为

$$dW(i, j)=lr*A(i)*(P(j)-W(i, j))$$

其中，lr 为学习速率，P 为神经元输入，A 为输出，W 为当前权值矩阵。

函数调用时各输入量和返回量的含义参见函数 learngd。该函数的学习参数为学习速率 LP.lr，默认值为 0.01。

【例 3-30】根据下列输入矢量 ***P***、输出矢量 ***A*** 和原始权值矩阵 ***W***，利用 learnis 函数对神经元权值进行学习。

```
P=[0.3578;0.1986;0.0135];
A=[0.7852;0.4423];
W=[0.6782 0.0198 0.3753;0.8217 0.6875 0.8631];
```

设置学习速率并更新权值。

```
LP.lr=0.6;
dW=learnis(W,P,[],[],A,[],[],[],[],[],LP,[])
dW =
    -0.1509     0.0842    -0.1705
    -0.1231    -0.1297    -0.2255
```

7. learnk

应用： Kohonen 权值学习函数。

格式： [dW, LS]=learnk(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

info=learnk(code)

解析： learnk 函数利用 Kohonen 规则对权值进行调整，该函数多用于竞争层网络中。Kohonen 规则可以使输入矢量存储于神经元权值中。在 learnk 函数中，当神经元输出为 0 时，权值不作调整；当输出不为 0 时，权值调整量为

$$dW(i, j)=lr*(P(j)-W(i, j))$$

其中，lr 为学习速率，P 为神经元输入，W 为当前权值矩阵。

函数调用时各输入量和返回量的含义参见函数 learngd。该函数的学习参数为学习速率 LP.lr，默认值为 0.01。

【例 3-31】对一个 3 输入两神经元的网络层，设定输入矢量 ***P*** 和权值矩阵 ***W***，计算输出矢量 ***A***。

```
P=[0.3578;0.1986;0.0135];
W=[0.6782 0.0198 0.3753;
   0.8217 0.6875 0.8631];
A=compet(negdist(W,P))
A =
   (1,1)          1
```

然后设置学习速率，并利用 Kohonen 学习规则更新权值。

```
LP.lr=0.6;
dW=learnk(W,P,[],[],A,[],[],[],[],[],LP,[])
dW =
    -0.1922     0.1073    -0.2171
          0          0          0
```

8. learnlv1

应用： LVQ1 权值学习函数。

格式： [dW, LS]=learnlv1(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

info=learnlv1(code)

解析： learnlv1 函数利用 LVQ1 规则对权值进行调整，该函数多用于学习矢量量化网络中。在 learnlv1 函数中，当神经元输出为 0 时，权值不作调整；当输出不为 0 时，权值将根据梯度矢量 gA 的取值进行调整。

$$dW(i, j)=lr*(P(j)-W(i, j)) \quad 当\ gA(i)=0$$

$$dW(i, j)=-lr*(P(j)-W(i, j)) \quad 当\ gA(i)\neq 0$$

其中，lr 为学习速率，P 为神元输入，W 为当前权值。

函数调用时各输入量和返回量的含义参见函数 learngd，其中梯度矢量 gA 用来表示该层神经元的分类结果是否正确。该函数的学习参数为学习速率 LP.lr，默认值为 0.01。

【例 3-32】根据输入矢量 ***P*** 和权值矩阵 ***W***，利用 learnlv1 函数对神经元权值进行学习。

```
P=[0.3578;0.1986;0.0135];
W=[0.6782 0.0198 0.3753;0.8217 0.6875 0.8631];
```

首先计算神经元的输出

```
A=compet(negdist(W,P));
```

设定梯度矢量

```
gA=[1;1];
```

设置学习速率并更新权值

```
LP.lr=0.5;
dW=learnlv1(W,P,[],[],A,[],[],[],gA,[],LP,[])
dW =
     0.1602    -0.0894     0.1809
          0          0          0
```

9．learnlv2

应用：LVQ2 权值学习函数。

格式：[dW, LS]=learnlv2(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

info=learnlv2(code)

解析：learnlv2 函数利用 LVQ2 规则对权值进行调整，该函数同样用于学习矢量量化网络中。但网络只有利用 LVQ1 规则进行学习后，才能使用 LVQ2。在利用 learnlv2 函数进行学习时，如果输出最大的两个神经元分别为 i 和 j，而且输入矢量落在这两个神经元的权值矢量中间，即满足

$$\min(di/dj, dj/di)>(1-window)/(1+window)$$

其中，di 和 dj 分别为输入矢量和这两个神经元权值矢量间的距离，window 为学习参数中的窗宽度参数。在上述条件下，如果神经元 i 输出为高时将产生正确的分类结果，神经元 j 输出为高时将产生错误的分类结果，那么这两个神经元的权值调整量为

$$dW(i,k)=lr*(P(k)-W(i,k))$$

$$dW(j, k)=-lr*(P(k)-W(i, k))$$

上式中 lr 为学习速率，P 为神经元输入，W 为当前权值矩阵。

函数调用时各输入量和返回量的含义参见函数 learngd，梯度矢量 gA 用来表示该层神经元的分类结果是否正确。该函数的学习参数包括如下。

- LP.lr：学习速率，默认值为 0.01；
- LP.window：窗宽度，取值可以是 0～1 间的任意实数，默认值为 0.25，通常取值在 0.2 和 0.3 之间。

【例 3-33】对于一个两输入 3 神经元的网络层，利用 learnlv2 函数进行学习，首先设定权值 ***W***、输入矢量 ***P***、梯度矢量 gA，并计算神经元的加权输入量 ***N*** 和输出量 ***A***。

```
W=[-0.75 1;1 0;1 1];
P=[0;1];
gA=[-1;1;1];
N=negdist(W,P);
A=compet(N);
```

然后设置学习速率并更新权值

```
LP.lr=0.5;
LP.window=0.25;
dW=learnlv2(W,P,[],N,A,[],[],[],gA,[],LP,[])
dW =
    -0.3750         0
          0         0
    -0.5000         0
```

10. learnos

应用： 星外权值学习函数。

格式： [dW, LS]=learnos(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

info=learnos(code)

解析： lernos 函数利用星外联想学习方法对权值进行调整，该方法中权值调整量为

dW(i, j)=lr*(A(i)-W(i, j))*P(j)

其中，lr 为学习速率，P 为神经元输入，A 为输出，W 为当前权值矩阵。

函数调用时各输入量和返回量的含义参见函数 learngd。该函数的学习参数为学习速率 LP.lr，默认值为 0.01。

【例 3-34】根据下列输入矢量 ***P***、输出矢量 ***A*** 和权值矩阵 ***W***，利用 learnos 函数对神经元权值进行学习。

```
P=[0.3578;0.1986;0.0135];
A=[0.7468;0.4451];
W=[0.6782 0.0198 0.3753;0.8217 0.6875 0.8631];
```

设置学习速率并更新权值

```
LP.lr=0.5;
dW=learnos(W,P,[],[],A,[],[],[],gA,[],LP,[])
dW =
     0.0123     0.0722     0.0025
    -0.0674    -0.0241    -0.0028
```

11. learnp

应用： 感知器权值和阈值学习函数。

格式： [dW, LS]=learnp(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

[dB, LS]=learnp(B, P, Z, N, A, T, E, gW, gA, D, LP, LS)

info=learnp(code)

解析： learnp 函数可用来对感知器网络的权值和阈值进行调整，该方法中权值调整是由输

入矢量 P 和误差矢量 E 决定。

dW(i, j)=E(i)*P(j)

函数调用时各参数的含义参见函数 learncon 和 learngd。该函数没有学习参数。

【例 3-35】一个 3 输入两神经元网络层的输入矢量 ***P*** 和误差矢量 ***E*** 如下。

```
P=[0.1597;0.6932;0.4521];
E=[1;-1];
```

使用 learnp 函数对权值进行更新

```
dW=learnp([],P,[],[],[],[],E,[],[],[],[],[])
dW =
     0.1597     0.6932     0.4521
    -0.1597    -0.6932    -0.4521
```

12. learnpn

应用： 归一化感知器权值和阈值学习函数。

格式： [dW, LS]=learnpn(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

[dB, LS]=learnpn(B, P, Z, N, A, T, E, gW, gA, D, LP, LS)

info=learnpn(code)

解析： learnpn 函数多用于感知器网络的权值和阈值调整，当输入矢量中存在奇异样本时，该方法可以有效地提高感知器网络的训练速度。在该函数中，权值调整量由归一化的输入矢量 P 和误差矢量 E 共同决定

dW(i, j)=E(i)*P(j)/nP

其中，nP 为输入矢量 P 的模值

nP=sqrt (1+P(1)^2+P(2)^2+⋯+P(R)^2)

函数各输入量和返回量的含义参见函数 learncon 和 learngd。该函数没有学习参数。

【例 3-36】一个 3 输入两神经元网络层的输入矢量 ***P*** 和误差矢量 ***E*** 如下。

```
P=[0.1597;0.6932;0.4521];
E=[1;-1];
```

利用 learnpn 函数对权值进行更新

```
dW=learnpn([],P,[],[],[],[],E,[],[],[],[],[])
dW =
     0.1221     0.5300     0.3457
    -0.1221    -0.5300    -0.3457
```

13. learnsom

应用： 自组织映射网络权值学习函数。

格式： [dW, LS]=learnsom (W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

info=learnsom (code)

解析： learnsom 函数用于自组织映射网络中神经元权值的学习。在 learnsom 函数中，不仅获胜神经元的权值要作调整，获胜神经元领域内的权值也要更新，即

dW(i, k)=lr*a2* (P(k)-W(i, k))

其中，lr 为学习速率，P 为神经元输入，W 为当前权值，a2 为活动系数。第 *i* 个神经元的活动系数 a2(i)按照如下方式确定。

a2(i)=1：如果第 *i* 个神经元获胜。

a2(i)=0.5：如果第 *j* 个神经元获胜，并且和第 *i* 个神经元的间距小于领域半径 nd。

a2(i)=0：其他。

函数调用时各输入量和返回量的含义参见函数 learngd。该函数的学习参数包括如下。

- LP.order_lr：排列阶段的学习速率，默认值为 0.9；
- LP.order_steps：排列阶段的学习次数，默认值为 1000；
- LP.tune_lr：调整阶段的学习速率，默认值为 0.02；
- LP.tune_nd：调整阶段的领域半径，默认值为 1。

在本函数中，学习分排列和调整两阶段进行，学习速率 lr 在两个阶段中要从 LP.order_lr 降低到 LP.tune_lr，领域半径 nd 要从神经元间的最大距离降低到 LP.tune_nd。

【例 3-37】构造一个单层两输入 9 神经元的自组织映射网络，网络的拓扑结构为六边形，神经元间距采用 linkdist 计算方法。

随机产生输入矢量 ***P***、权值矩阵 ***W*** 和输出矢量 ***A***。

```
Pos=hextop(3,3);
%绘制神经元拓扑结构图，如图 3-17 所示。
plotsom(Pos);
```

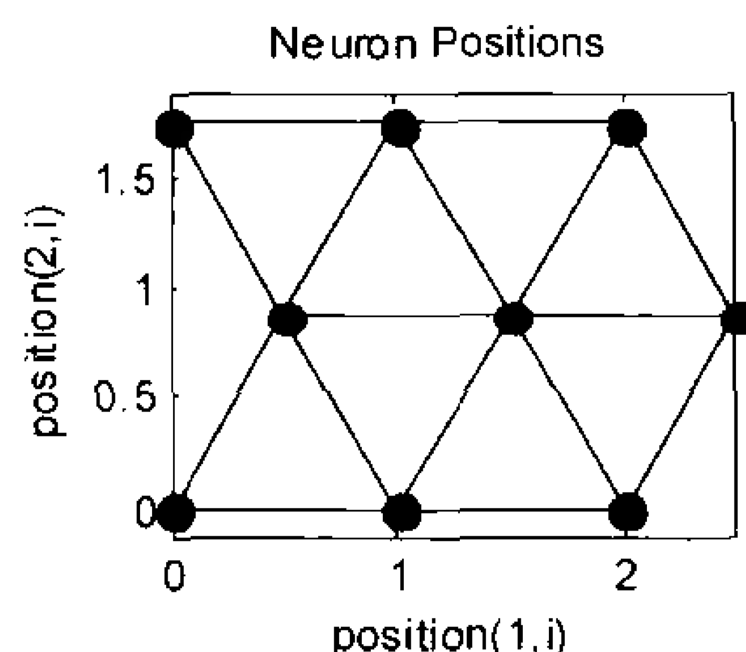


图 3-17　神经元拓扑结构图

```
D=linkdist(Pos);
P=rand(2,1);
W=rand(9,2);
%绘制原始神经元权值矢量，如图 3-18 所示。
plotsom(W,D);
A=rand(9,1);
```

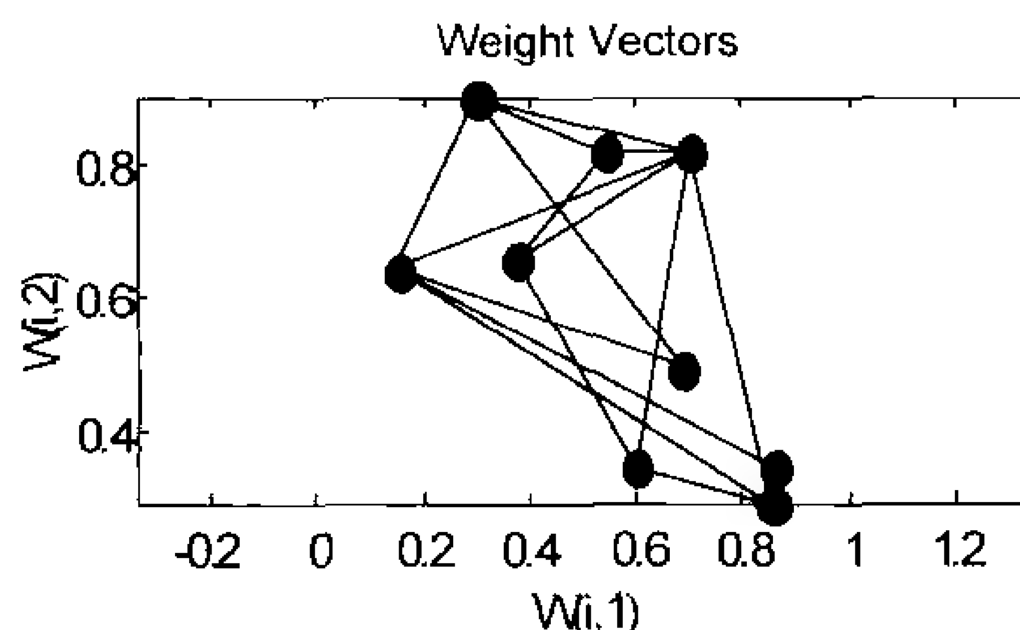


图 3-18　原始神经元权值矢量

初始学习状态设为[]，学习速率取为默认值，并更新权值

```
LP.order_lr=0.9;
```

```
LP.order_steps=1000;
LP.tune_lr=0.02;
LP.tune_nd=1;
LS=[];
[dW,LS]=learnsom(W,P,[],[],A,[],[],[],[],D,LP,LS);
%绘制更新后的神经元权值矢量，如图 3-19 所示。
plotsom(W+dW,D);
```

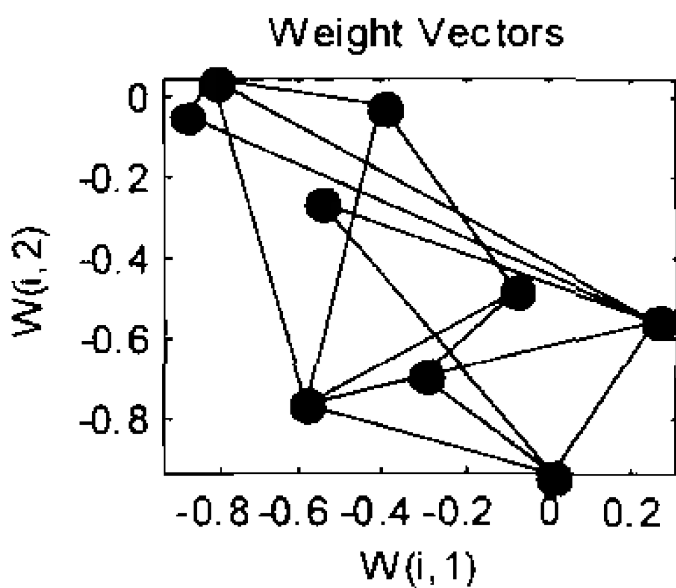


图 3-19　更新后神经元权值矢量

14. learnwh

应用： Widrow-Hoff 权值和阈值学习函数。

格式： [dW, LS]=learnwh (W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

[dB, LS]=learnwh (B, P, Z, N, A, T, E, gW, gA, D, LP, LS)

info=learnwh (code)

解析： learnwh 利用 Widrow-Hoff 规则对网络的权值和阈值进行学习，该函数多用于线性网络。在该方法中权值调整量由输入矢量 P、误差矢量 E 和学习速率 lr 共同确定。

$$dW(i, j)=lr*E(i)*P(j)$$

函数调用时各输入量和返回量的含义参见函数 learncon 和 learngd。该函数学习参数是由学习速率 LP.lr 构成的，默认值为 0.01。

【例 3-38】构造一个 3 输入两神经元网络层，输入矢量 ***P*** 和误差矢量 ***E*** 如下。

```
P=[0.1597;0.6932;0.4521];
E=[0.3789;0.7862];
```

设定学习速率，并利用 learnwh 函数对权值进行学习

```
LP.lr=0.6;
dW=learnwh([],P,[],[],[],[],E,[],[],[],LP,[])
dW =
     0.0363     0.1576     0.1028
     0.0753     0.3270     0.2133
```

3.6　神经网络的输入函数及其导函数

神经网络的输入函数及其导函数如表 3-6 所示。

表 3-6　神经网络的输入函数及其导数

函数名称	功　能	函数名称	功　能
netprod	乘积输入函数	dnetprod	乘积输入函数的导函数
netsum	求和输入函数	dnetsum	求和输入函数的导函数

3.6.1 输入函数

1．netprod

应用：乘积输入函数。

格式：N=netprod (Z1, Z2,⋯, Zn)

df=netprod ('deriv')

解析：netprod 函数以乘积方式把加权输入和阈值组合在一起。newpnn 和 newgrnn 函数产生的网络以 netprod 函数作为输入函数。

在函数调用 N=netprod (Z1, Z2,⋯, Zn)中，Z1 到 Zn 分别为加权输入矩阵或阈值矩阵，此时函数把 Z1 到 Zn 各矩阵中同一位置上的元素分别相乘，然后返回一个维数相同的输入矩阵。函数的调用形式 df=netprod ('deriv')返回 netprod 函数的导函数的名称，即'dnetprod'。

【例 3-39】设定加权输入矩阵 Z1、Z2 和阈值矢量 ***b***，计算其乘积输入。

```
Z1=[1 4;4 6];
Z2=[-5 2;-3 1];
b=[0;-2];
```

为处理批处理数据，需要把阈值矢量扩展为维数相同的矩阵

```
B=concur(b,2)
N=netprod(Z1,Z2,B)
B =
     0     0
    -2    -2
N =
     0     0
    24   -12
```

2．netsum

应用：求和输入函数。

格式：N=netsum (Z1, Z2,⋯, Zn)

df=netsum ('deriv')

解析：netsum 函数以求和方式把加权输入和阈值组合在一起。newp 和 newlin 等函数产生的网络以 netsum 函数作为输入函数。

在函数的调用形式 N=netsum (Z1, Z2,⋯, Zn)中，Z1 和 Zn 分别为加权输入矩阵或阈值矩阵，此时函数把 Z1 到 Zn 各矩阵中同一位置上的元素分别相加，然后返回一个维数相同的输入矩阵。函数的调用形式 df=netsum ('deriv')返回 netsum 函数的导数函数的名称，即'dnetsum'。

【例 3-40】设定加权输入矩阵 Z1、Z2 和阈值矢量 ***b***，计算其求和输入。

```
Z1=[1 4;4 6];
Z2=[-5 2;-3 1];
b=[0;-2];
```

为处理批处理数据，需要把阈值矢量扩展为维数相同的矩阵

```
B=concur(b,2);
N=netsum(Z1,Z2,B)
```

```
N =
      -4      6
      -1      5
```

3.6.2　输入函数的导函数

1．dnetprod

应用：乘积输入函数的导函数。

格式：dN_dZ=dnetprod (Z, N)

解析：dnetprod 函数用来求取乘积输入函数的输出对加权输入或阈值的导数。

在调用函数中，Z 为加权输入矩阵或阈值矩阵，N 为乘积输入函数的输出，导函数返回输入函数的输出 N 对加权输入或阈值 Z 的导数矩阵。

【例 3-41】设定加权输入矩阵 Z1、Z2 和阈值矢量 ***b***，计算乘积输入值及其对 Z1 的导数。

```
Z1=[3 4;2 7];
Z2=[-3 2;4 8];
b=[0;-1];
B=concur(b,2);
N=netprod(Z1,Z2,B);
dN_dZ1=dnetprod(Z1,N)
dN_dZ1 =
      0      0
     -4     -8
```

2．dnetsum

应用：求和输入函数的导函数。

格式：dN_dZ=dnetsum (Z, N)

解析：dnetsum 函数用来计算求和输入函数的输出 N 对加权输入或阈值 Z 的导数。函数调用时各输入量和返回量的含义参见 dnetprod 函数。

【例 3-42】设定加权输入矩阵 Z1、Z2 和阈值矢量 ***b***，计算求和输入值及其对 Z1 的导数。

```
Z1=[3 4;2 7];
Z2=[-3 2;4 8];
b=[0;-1];
B=concur(b,2);
N=netsum(Z1,Z2,B);
dN_dZ1=dnetsum(Z1,N)
dN_dZ1 =
      1      1
      1      1
```

3.7　神经网络的性能函数及其导函数

神经网络的性能函数及其导函数如表 3-7 所示。性能函数通过计算网络输出矢量和目标矢量之间的差异来衡量神经网络的性能；而导函数是用来求取性能函数的输出对于各权值及阈值参数的导数，在程序中不直接调用。

表 3-7　神经网络的性能函数及其导数

函数名称	功　能	函数名称	功　能
mae	平均绝对误差函数	dmae	平均绝对误差函数的导函数
mse	均方误差函数	dmse	均方误差函数的导函数
msereg	规则化均方误差函数	dmsereg	规则化均方误差函数的导函数
sse	平方和误差函数	dsse	平方和误差函数的导函数

3.7.1　性能函数

1．mae

应用： 平均绝对误差函数。

格式： perf=mae (E, x, pp)

perf=mae (E, net, pp)

info=mae (code)

解析： E 为误差向量矩阵（或向量）；x 为所有的权值和阈值向量，可忽略；pp 为性能参数，可忽略；net 为待评定的神经网络；perf 表示平均绝对误差和；mae(code)为根据不同的 code 值返回不同的信息，包括如下。

deriv：返回导数函数的名称；

name：返回函数的全称；

pnames：返回训练参数的名称；

pdefaults：返回默认的训练参数。

【例 3-43】感知器最重要的也是最实用的功能是对输入向量进行分类。本例尝试建立一个感知器模型，实现电路“或”门的功能，从而实现对输入的分类。

“或”门网络的输入向量是 ***P*** 和目标向量是 ***T***。其中，***P***=[0 0 1 1;0 1 0 1]；***T***=[0 1 1 1]。

本例的 MATLAB 代码为

```
P=[0 0 1 1;0 1 0 1];
T=[0 1 1 1];
net=newp(minmax(P),1);
Y1=sim(net,P)
net.trainParam.epochs=25;
net=train(net,P,T);
Y2=sim(net,P)
perf=mae(Y2-T)
```

输出为

```
Y1 =
     1     1     1     1
TRAINC, Epoch 0/25
TRAINC, Epoch 4/25
TRAINC, Performance goal met.
Y2 =
     0     1     1     1
perf =
     0
```

由此可见，感知器在训练以前的输出是不符合要求的，经过 4 次训练后的输出已经和目标矢量一致了，训练过程的误差曲线如图 3-20 所示。

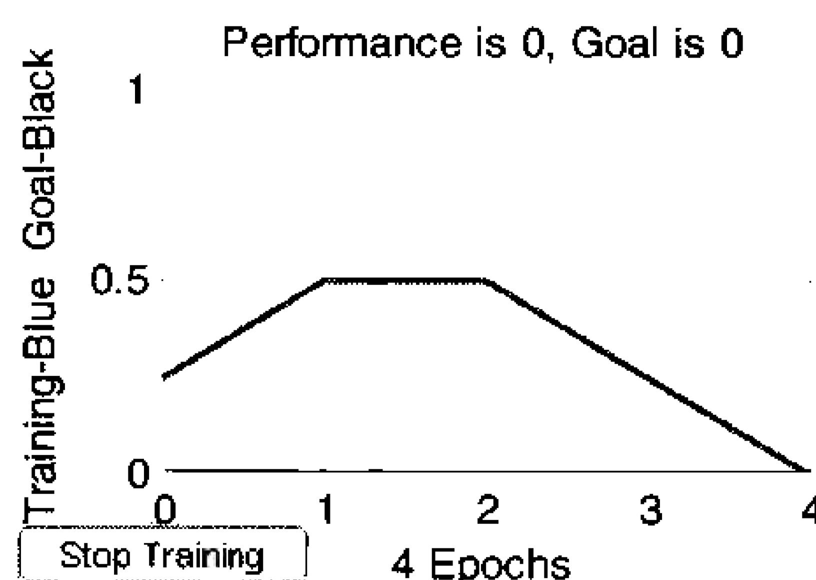


图 3-20　训练过程的误差曲线

本例创建的感知器只有一个神经元，这也符合了神经网络的设计原则，即在设计神经网络时，在同样可以完成任务的情况下，尽量采用简单的结构。因为结构简单的神经网络计算负担轻，运行速度一般比较快。感知器的传递函数和学习函数都采用默认值，分别为 hardlim 和 learnp，这是因为网络的输出为 0-1 的二值结构，只有采用 hardlim 才满足要求。输入向量中不存在奇异值，元素之间的距离也比较小，因此，采用 learnp 就足够了。

2. mse

应用：均方误差函数。

格式：perf=mse (E, x, pp)

perf=mse (E, net, pp)

info=mse (code)

说明：

mse 函数用来计算输出矢量和目标矢量之间的均方误差。函数调用时各输入量和返回量的含义参见 mae 函数。利用 newcf、newff 和 newelm 函数建立的网络可以采用 mse 函数作为其性能函数。

【例 3-44】计算下面误差矢量的均方误差。

```
E=[0.2698;-1.3682;1.8523];
mse(E)
```

输出结果为

```
ans =
    1.7919
```

3. msereg

应用：规则化均方误差函数。

格式：perf=msereg (E, x, pp)

perf=msereg (E, net, pp)

info=msereg (code)

解析：msereg 函数用来计算网络的规则化均方误差，该性能函数不仅考虑了网络输出的均方误差，还考虑了网络中各权值和阈值参数的均方和，网络的性能由这两者的加权和决定。函数调用时各输入量和返回量的含义参见 mae 函数。该函数的性能参数如下。

pp.ratio：误差函数中网络输出均方误差的权重因子，1-pp.ratio 则是权值和阈值参数均方和的权重因子。

由于 msereg 函数在调用时要使用网络参数和性能参数，因此函数输入 x、net 以及 pp 不能省略。

利用 newcf、newff 和 newelm 函数建立的网络可以采用 msereg 函数作为其性能函数。

【例 3-45】创建一个 BP 网络，并评估其性能。

```
%创建一个 BP 网络
net=newff([-2 2],[4 1],{'tansig','purelin'},'trainlm','learngdm','msereg');
P=[-3 -2 0 2 3];
T=[0 1 1 1 0];
Y=sim(net,P)
%误差向量
E=T-Y
%设置性能参数
net.performparam.ratio=25/(25+1);
perf=msereg(E,net)
```

输出结果为

```
Y =
    1.2957    0.8929    -0.0122    -0.4355    -1.1074
E =
   -1.2957    0.1071    1.0122    1.4355    1.1074
perf =
    1.4553
```

4．sse

应用： 平方和误差函数。

格式： perf=sse (E, x, pp)

perf=sse (E, net, pp)

info=sse (code)

解析： sse 函数用来计算网络输出矢量和目标矢量之间的误差平方和。函数调用时各输入量和返回量的含义参见 mae 函数。

【例 3-46】对下面的误差矢量计算平方和误差。

```
E=[0.2698;-1.3682;1.8523];
sse (E)
```

输出为

```
ans =
    5.3758
```

3.7.2 性能函数的导函数

1．dmae

应用： 平均绝对误差函数的导函数。

格式： dperf_dE=dmae ('e', E, X, perf, pp)

dperf_dX=dmae('x', E, X, perf, pp)

解析： dmae 函数用来计算平均绝对误差函数的输出对其输入的导数。

在函数的调用形式中，输入 E 为误差矢量；X 是由网络权值和阈值参数构成的矩阵；perf 为网络的性能值；pp 为性能参数。调用形式 dperf_dE=dmae ('e', E, X, perf, pp)返回网络性能对于网络输出误差的导数 dperf_dE，调用形式 dperf_dX=dmae('x', E, X, perf, pp)返回网络性能对于各权值和阈值的导数 dperf_dX。

【例 3-47】设网络的输出误差矢量 ***E*** 和权值矢量 ***X*** 为

```
E=[1;-0.5];
X=[3.5;0.2;-2.2;4.1];
```

首先计算平均绝对误差

```
perf=mae(E)
perf =
    0.7500
```

然后计算平均绝对误差对网络输出误差和权值的导数

```
dperf_dE=dmae('e',E,X)
dperf_dE =
        [2x1 double]
dperf_dX=dmae('x',E,X);
dperf_dX' =
ans =
     0     0     0     0
```

2．dmse

应用：均方误差函数的导函数。

格式：dperf_dE=dmse ('e', E, X, perf, pp)

　　　dperf_dX=dmse ('x', E, X, perf, pp)

解析：dmse 函数用来计算均方误差函数的输出对其输入的导数。函数调用时各输入量和返回量的含义参见 dmae 函数。

【例 3-48】根据下列误差矢量 ***E*** 和权值矢量 ***X*** 计算网络的均方误差及其导数。

```
E=[1.2;-0.6];
X=[3.5;0.2;-2.2;4.3];
perf=mae(E)
perf =
    0.6250
dperf_dE=dmse('e',E,X)
dperf_dE =
        [2x1 double]
dperf_dX=dmse('x',E,X);
dperf_dX=dmae('x',E,X);
dperf_dX' =
ans =
     0     0     0     0
```

3．dmsereg

应用：规则化均方误差函数的导函数。

格式：dperf_dE=dmsereg ('e', E, X, perf, pp)

dperf_dX=dmsereg ('x', E, X, perf, pp)

解析：dmsereg 函数用来计算规则化均方误差函数的输出对其输入的导数。性能参数 pp 参见函数 msereg，函数调用时其余各输入量和返回量的含义参见 dmae 函数。

【例 3-49】对下列误差矢量 ***E*** 和权值矢量 ***X*** 计算规则化均方误差及其导数。

```
E=[11;-0.5];
X=[3.5;0.2;-2.2;4.1];
pp.ratio=0.9;
perf=msereg(E,X,pp)
perf =
     1.4110
dperf_dE=dmsereg('e',E,X,perf,pp)
dperf_dE =
     0.9000
    -0.4500
dperf_dX=dmsereg('x',E,X,perf,pp);
dperf_dX'
ans =
    -0.1750    -0.0100     0.1100    -0.2050
```

4. dsse

应用：平方和误差函数的导函数。

格式：dperf_dE=dsse ('e', E, X, perf, pp)

dperf_dX=dsse ('x', E, X, perf, pp)

解析：dsse 函数用来计算平方和误差函数的输出对其输入的导数。函数调用时各参数的含义参见 dmae 函数。

【例 3-50】对下列误差矢量 ***E*** 和权值矢量 ***X*** 计算平方和误差及其导数。

```
E=[1;-0.5];
X=[3.5;0.2;-2.2;4.1];
perf=sse(E,X,pp)
perf =
     1.2500
dperf_dE=dsse('e',E,X,perf,pp)
dperf_dE =
      2
     -1
dperf_dX=dmsereg('x',E,X,perf,pp);
dperf_dX'
ans =
    -0.1750    -0.0100     0.1100    -0.2050
```

3.8 传递函数及其导函数

神经网络的传递函数及其导函数如表 3-8 所示。

表 3-8　神经网络的传递函数及其导数

函数名称	功　　能	函数名称	功　　能
compet	竞争传递函数	—	—
hardlim	硬限幅传递函数	dhardlim	硬限幅传递函数的导函数
hardlims	对称硬限幅传递函数	dhardlims	对称硬限幅传递函数的导函数
logsig	对数 Sigmoid 传递函数	dlogsig	对数 Sigmoid 传递函数的导函数
poslin	正线性传递函数	dposlin	正线性传递函数的导函数
purelin	纯线性传递函数	dpurelin	纯线性传递函数的导函数
radbas	高斯径向基传递函数	dradbas	高斯径向基传递函数的导函数
satlin	饱和线性传递函数	dsatlin	饱和线性传递函数的导函数
satlins	对称饱和线性传递函数	dsatlins	对称饱和线性传递函数的导函数
softmax	softmax 传递函数	—	—
tansig	正切 Sigmoid 传递函数	dtansig	正切 tansig 传递函数的导函数
tribas	三角基传递函数	dtribas	三角基传递函数的导函数

3.8.1　传递函数

1．compet

应用：竞争传递函数。

格式：A=compet (N)

info=compet (code)

解析：竞争传递函数 compet 用来寻找输入矢量 N 中的最大元素，并把相应神经元的输出设为 1，其余输出设为 0。输出为 1 的神经元称为获胜神经元。

函数调用形式 info=compet (code)返回有关的函数信息，code 字符串取值如下。

'deriv'：返回导函数的名称，函数 compet 没有导函数；

'name'：返回传递函数的全称；

'output'：返回传递函数的输出范围；

'active'：返回传递函数的活跃输出范围。

newc 和 newpnn 函数建立的网络都采用 compet 函数作为传递函数。

【例 3-51】 下面给出一组 MATLAB 代码，可以演示该函数的功能和运行机理。

```
N=[0;1;-0.5;0.5];
A=compet(N)
info=compet('name')
subplot(2,1,1)
bar(N),ylabel('N')
subplot(2,1,2)
bar(A),ylabel('A')
```

向量 $\boldsymbol{A}$ 和 $\boldsymbol{N}$ 如图 3-21 所示。

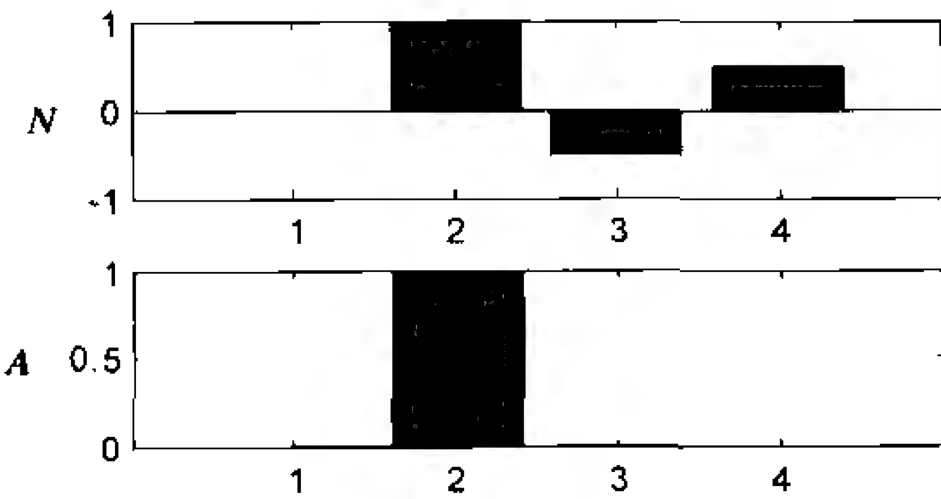


图 3-21　函数 compet 的向量 $\boldsymbol{A}$ 和 $\boldsymbol{N}$

运行结果为

```
A =
   (2,1)        1
info =
Competitive
```

2. softmax

应用： softmax 传递函数。

格式： A=softmax (N)

info=softmax (code)

解析： softmax 传递函数的计算公式用 MATLAB 语句表示为

a=exp(n)/sum(exp(n))

其参数含义参见 compet 函数，与 compet 不同的是，参数 A 为函数返回向量，各元素在区间[0，1]之间，且向量结构与 N 一致。该函数也没有导函数。

【例 3-52】利用如下一组代码演示该函数的功能和运行机理。

```
N=[0;1;-0.5;0.5];
A=softmax(N)
info=softmax('name')
subplot(2,1,1)
bar(N),ylabel('N')
subplot(2,1,2)
bar(A),ylabel('A')
```

运行结果为

```
A =
     0.1674
     0.4551
     0.1015
     0.2760
info =
Soft Max
```

向量 ***A*** 和 ***N*** 如图 3-22 所示。

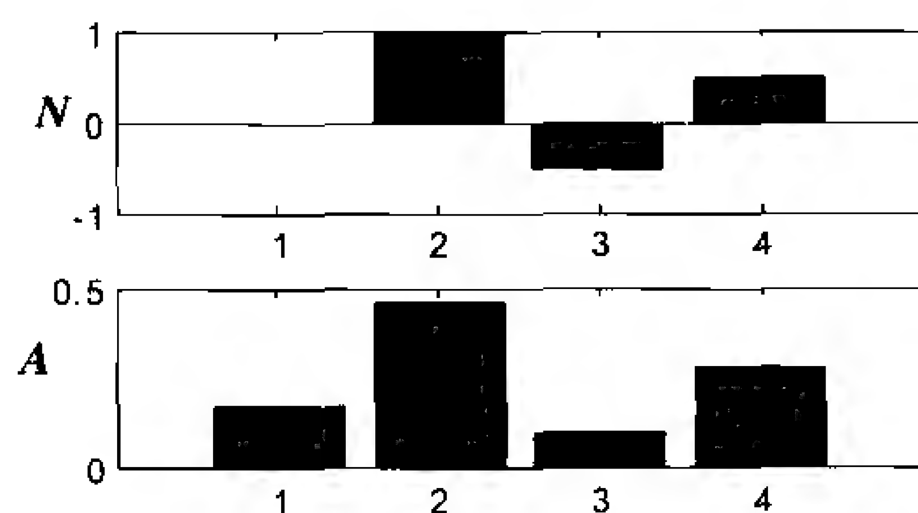


图 3-22　函数 softmax 的向量 ***A*** 和 ***N***

3. hardlim

应用： 硬限幅传递函数。

格式： A=hardlim (N)

info=hardlim (code)

解析： 硬限幅传递函数 hardlim 把函数输入矢量 N 中各元素的取值强迫限制为 1 或 0，当

输入大于或等于 0 时，神经元的输出为 1，否则输出为-1。

newp 函数建立的网络可以采用 hardlims 函数作为传递函数。

该函数的原型函数为

$$\text{hardlim}(n)=\begin{cases}1 & n\geqslant 0\\ 0 & n<0\end{cases}$$

4. hardlims

应用： 对称硬限幅传递函数。

格式： A=hardlims (N)

info=hardlims (code)

解析： 对称硬限幅传递函数 hardlims 把函数输入矢量 N 中各元素的取值强迫限制为 1 或-1，当输入大于或等于 0 时，神经元的输出为 1，否则输出为 0。

newp 函数建立的网络可以采用 hardlim 函数作为传递函数。

该函数的原型函数为

$$\text{hardlims}(n)=\begin{cases}1 & n\geqslant 0\\ -1 & n<0\end{cases}$$

由此可见，以上两个函数可以实现神经网络的分类和判断功能。

【例 3-53】利用 MATLAB 给出一个数组，调用函数 hardlim 与 hardlims 对其进行分类。MATLAB 代码如下。

```
%以 0.1 为步长，建立一个数组
N=-6:0.1:6;
A1=hardlim(N);
A2=hardlims(N);
plot(N,A1,'ro');
hold on
plot(N,A2,'b+');
hold on
```

运行结果如图 3-23 所示。

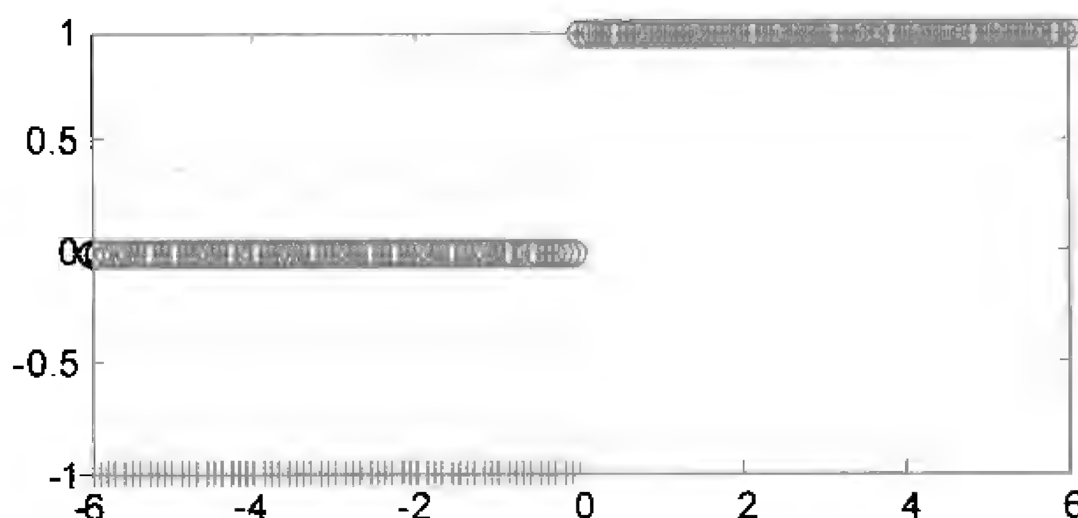


图 3-23　利用传递函数进行分类的结果

图中加号部分是函数 hardlims 的分类结果；圆圈部分是 hardlim 的分类结果。可以看出两个函数都成功地对数组进行了分类。

5．logsig

应用： 对数 Sigmoid 传递函数。

格式： A=logsig (N)

info=logsig (code)

解析：对数 Sigmoid 传递函数的计算公式用 MATLAB 语句表示为

logsig(n)=1/(1+exp(-n))

newff 和 newcf 函数建立的网络都可以采用该函数作为传递函数。

【例 3-54】下面以一组简单的代码产生一个对数 S 型的传递函数为例。

```
N=-10:0.1:10;
A=logsig(N)
plot(N,A)
grid
```

运行结果如图 3-24 所示。可见，函数 logsig 可将神经元的输入（范围为整个实数集）映射到区间［0，1］中，又由于该函数为可微函数，因此非常适合于利用 BP 算法训练神经网络。

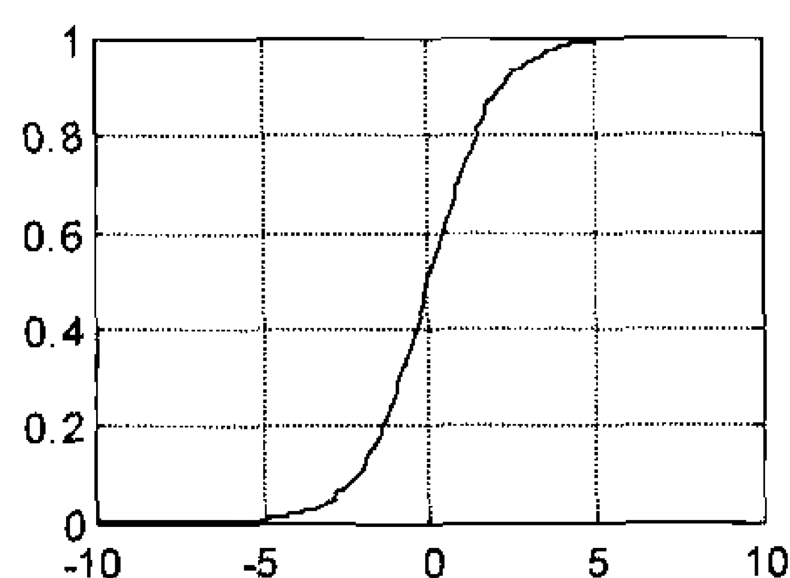


图 3-24　logsig 函数运行结果

6．poslin

应用：正线性传递函数。

格式：A=poslin (N)

　　info=poslin (code)

解析：对于输入矢量 N 中的正元素或 0，正线性传递函数按照原始数值输出；对于负元素，函数将输出 0。

【例 3-55】求正线性传递函数对下列输入矢量的输出。

```
N=[-2;0;-1;3];
A=poslin(N);
A'
plot(N,A)
```

运行结果如下，如图 3-25 所示。

```
ans =
     0     0     0     3
```

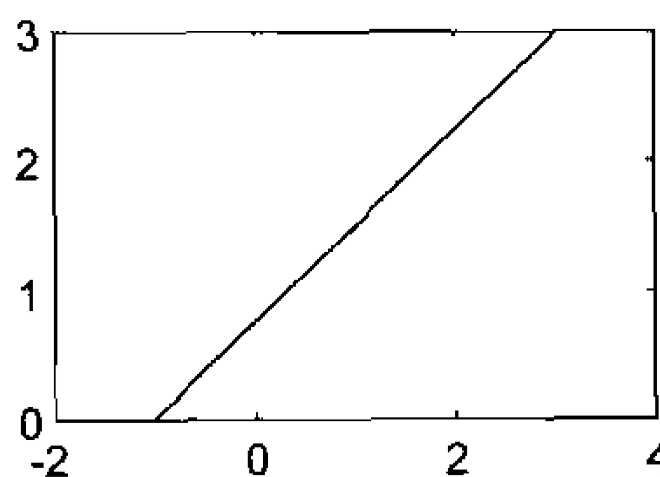


图 3-25　poslin 的运行结果

7．purelin

应用：纯线性传递函数。

格式： A=purelin (N)

info=purelin (code)

解析： 纯线性传递函数把输入 N 按照原始数据输出。newlin 和 newlind 函数建立的网络都可以采用该函数作为传递函数。

【例 3-56】下面以一组简单的代码产生一个线性 S 型传递函数。

```
N=-10:0.1:10;
A=purelin(N);
plot(N,A)
grid
```

运行结果如图 3-26 所示。

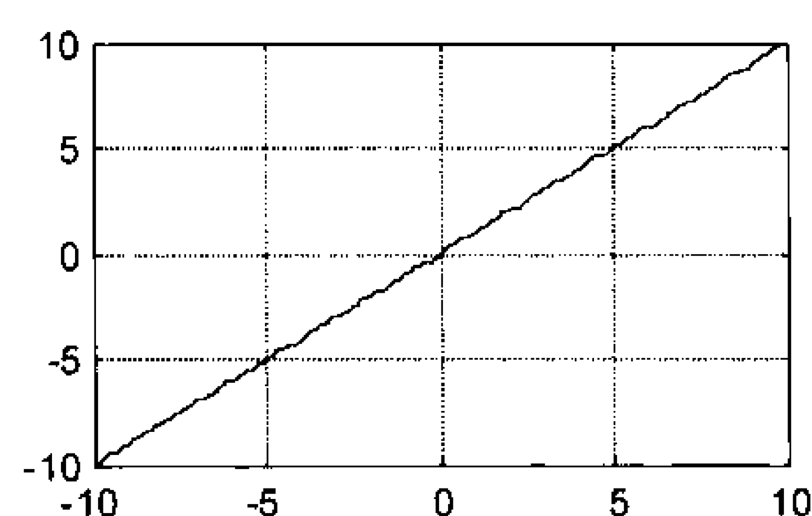


图 3-26　purelin 函数运行结果

8．radbas

应用： 高斯径向基传递函数。

格式： A=radbas (N)

info=radbas (code)

解析： 高斯径向基传递函数的计算公式用 MATLAB 语句表示为

a=exp(-n.^2)

newpnn 和 newgrann 函数建立的网络都可以采用该函数作为传递函数。

【例 3-57】下面以一组简单的代码产生一个高斯径向基传递函数。

```
N=-10:0.1:10;
A=radbas(N);
plot(N,A)
grid
```

运行结果如图 3-27 所示。

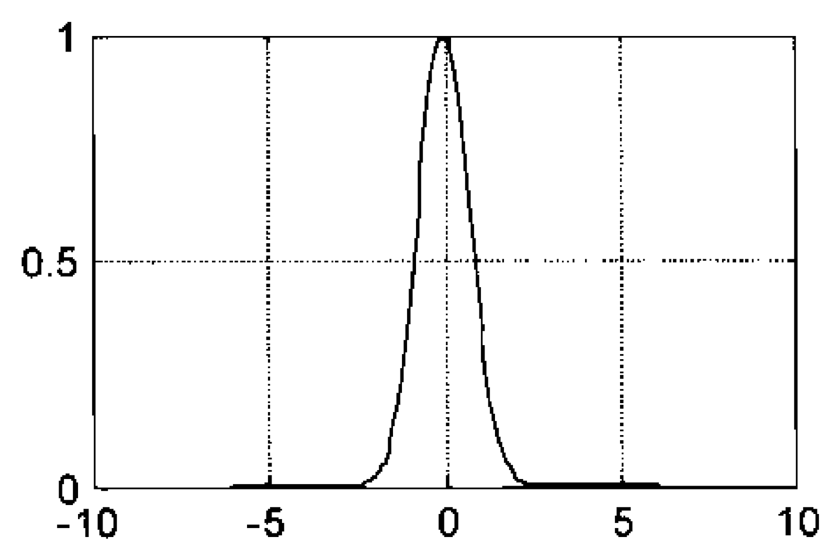


图 3-27　radbas 函数运行结果

9．satlin

应用： 饱和线性传递函数。

格式： A=satlin (N)

info=satlin (code)

解析：饱和线性传递函数把输入矢量 N 中位于[0，1]区间内的元素按照原始数据输出；对于小于 0 的元素，则输出 0；对于大于 1 的元素，则输出 1。

【例 3-58】求饱和线性传递函数对下列输入矢量的输出。

```
N=[-3;-1;0.5;1;3];
A=satlin(N);
A'
ans =
        0         0    0.5000    1.0000    1.0000
```

10．satlins

应用：对称饱和线性传递函数。

格式：A=satlins (N)

info=satlins (code)

解析：对称饱和线性传递函数把输入矢量 N 中位于[−1，1]区间内的元素按照原始数据输出；对于小于−1 的元素，则输出−1；对于大于 1 的元素，则输出 1。

newhop 函数建立的网络采用该函数作为传递函数。

【例 3-59】求对称饱和线性传递函数对下列输入矢量的输出。

```
N=[-3;-1;0.5;1;3];
A=satlins(N);
A'
ans =
   -1.0000   -1.0000    0.5000    1.0000    1.0000
```

11．tansig

应用：正切 Sigmoid 传递函数。

格式：A=tansig (N)

info=tansig (code)

解析：tansig 传递函数的计算公式用 MATLAB 语句表示为

n=2/(1+exp (-2*n))-1

newff 和 newcf 函数建立的网络都可以采用该函数作为传递函数。

【例 3-60】以下代码可绘制一个双曲正切 S 型传递函数。

```
N=-10:0.1:10;
A=tansig(N);
plot(N,A)
grid
```

运行结果如图 3-28 所示。

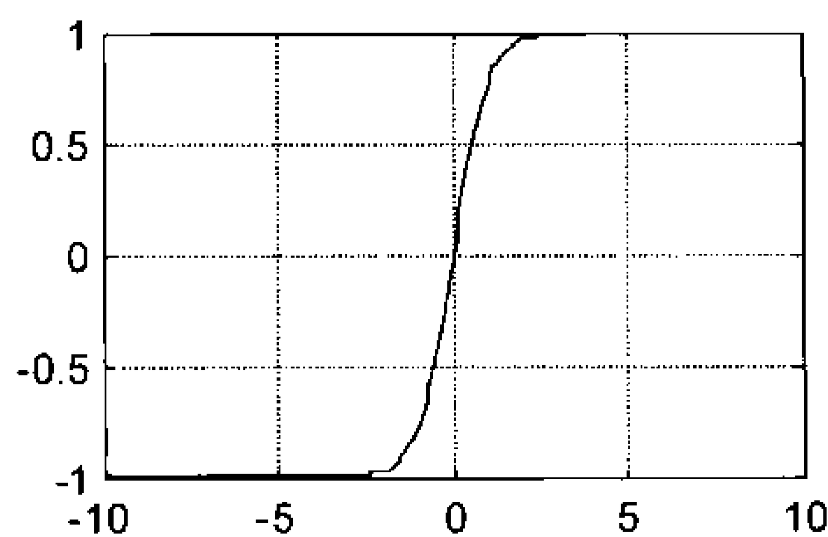


图 3-28　tangsig 函数运行结果

12．tribas

应用： 三角基传递函数。

格式： A=tribas (N)

info=tribas (code)

解析： 当输入矢量 N 中的元素位于[-1，1]区间时，tribas 传递函数的计算公式用 MATLAB 语句表示为

$$n=1-abs(n)$$

当元素在该区间外时，函数输出为 0。

【例 3-61】求 tribas 传递函数对下列输入矢量的输出。

```
N=[-3;-1;-0.8;0.5;0.8;1;3];
A=tribas(N);
A'
ans =
         0         0    0.2000    0.5000    0.2000         0         0
```

3.8.2　传递函数的导函数

1．dhardlim

应用： 硬限幅传递函数的导函数。

格式： dA_dN=dhardlim (N, A)

解析： 该函数用来计算硬限幅函数的输出 A 相对于输入 N 的导数。

【例 3-62】利用 MATLAB 给出一个数组，调用函数 dhardlim 对其进行分类。MATLAB 代码如下。

```
%以 0.1 为步长，建立一个数组
N=-6:0.1:6;
A1=hardlim(N);
dA_dN=dhardlim(N,A1)
plot(N,A,'r+');
hold on
```

运行效果如图 3-29 所示。

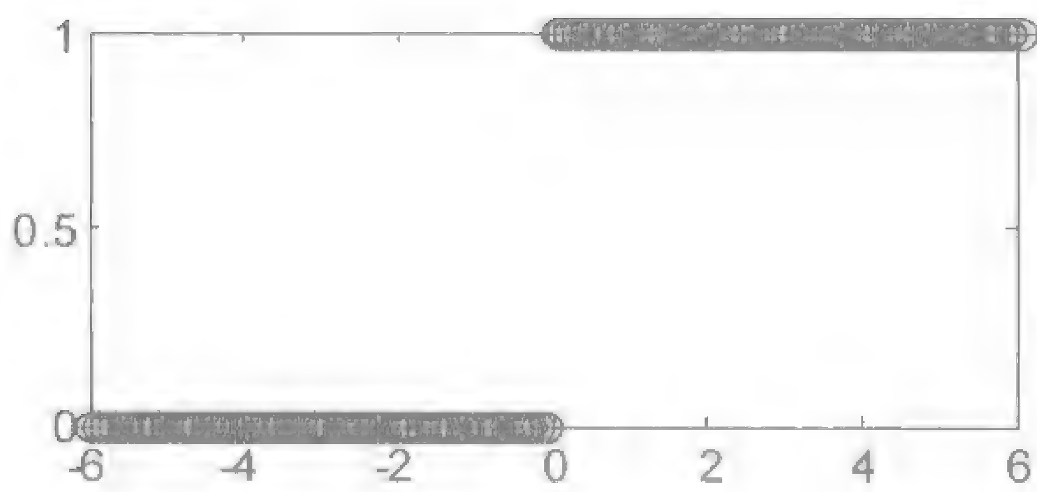


图 3-29　dhardlim 函数运行结果

2．dhardlims

应用： 对称硬限幅传递函数的导函数。

格式： dA_dN=dhardlims (N, A)

解析： 该函数用于计算对称硬限幅函数的输出 A 相对于输入 N 的导数。

【例 3-63】求对称硬限幅函数的输出 *A* 相对于输入 *N* 的导数。

```
N=[-3;-1;-0.8;0.5;0.8;1;3];
A=hardlims(N);
dA_dN=dhardlim(N,A);
dA_dN'
ans =
     0     0     0     0     0     0     0
```

3. dlogsig

应用： 对数 Sigmoid 传递函数的导函数。

格式： dA_dN=dlogsig (N, A)

解析： 该函数用于计算对数 Sigmoid 函数的输出 A 相对于输入 N 的导数。

【例 3-64】现有某 BP 网络层的 3 个神经元，其传递函数均为 S 型的对数函数，试利用函数 dlogsig 求解输出对输入的导数。

```
N=[-3;-1;-0.8;0.5;0.8;1;3];
N=[0.2;0.9;-0.6;-2];
A=logsig(N)
dA_dN=dlogsig(N,A)
```

运行结果如下

```
A =
    0.5498
    0.7109
    0.3543
    0.1192
dA_dN =
    0.6791
    0.8335
    0.4389
    0.1323
```

4. dposlin

应用： 正线性传递函数的导函数。

格式： dA_dN=dposlin (N, A)

解析： 该函数用于计算正线性函数的输出 A 相对于输入 N 的导数。

【例 3-65】求正线性传递函数的输出对下列输入矢量的导数。

```
N=[-1;0;0.5];
A=poslin(N);
dA_dN=dposlin(N,A)
dA_dN =
     0
     1
     1
```

5. dpurelin

应用： 纯线性传递函数的导函数。

格式： dA_dN=dpurelin (N, A)

解析： 该函数用于计算纯线性函数的输出 A 相对于输入 N 的导数。

【例 3-66】求纯线性传递函数的输出对下列输入矢量的导数。

```
N=[-1;0;0.5];
A=purelin(N);
dA_dN=dpurelin(N,A)
dA_dN =
      1
      1
      1
```

6．dradbas

应用： 高斯径向基传递函数的导函数。

格式： dA_dN=dradbas (N, A)

解析： 该函数用于计算高斯径向基函数的输出 A 相对于输入 N 的导数。

【例 3-67】下面以一组简单的代码产生一个高斯径向基传递函数的导函数。

```
N=-10:0.1:10;
A=radbas(N);
dA_dN=dradbas(N,A);
plot(N,dA_dN)
grid
```

运行结果如图 3-30 所示。

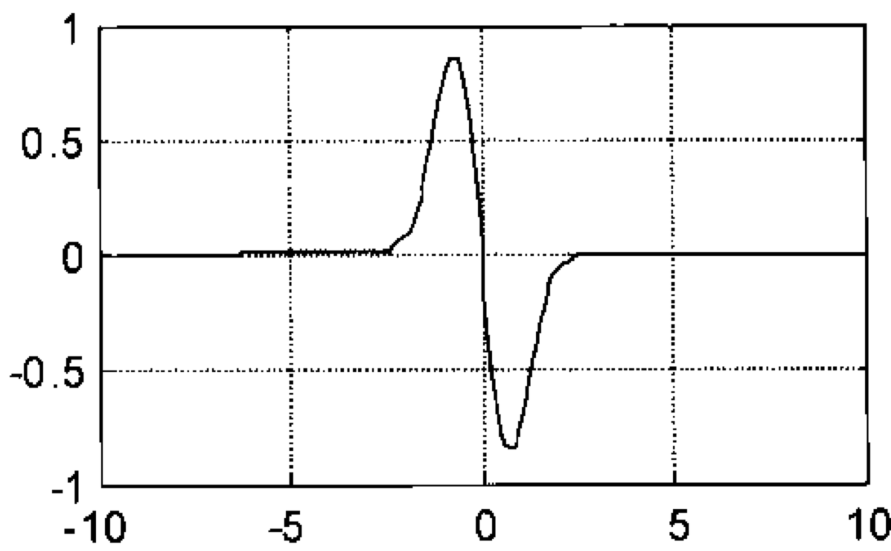


图 3-30　dradbas 函数运行结果

7．dsatlin

应用： 饱和线性传递函数的导函数。

格式： dA_dN=dsatlin (N, A)

解析： 该函数用于计算饱和线性函数的输出 A 相对于输入 N 的导数。

【例 3-68】求饱和线性传递函数的输出对下列输入矢量的导数。

```
N=[-1;0;0.5;2];
A=satlin(N);
dA_dN=dsatlin(N,A)
dA_dN =
      0
      1
      1
      0
```

8．dsatlins

应用： 对称饱和线性传递函数的导函数。

格式：dA_dN=dsatlins (N, A)

解析：该函数用于对称饱和线性函数的输出 A 相对于输入 N 的导数。

【例 3-69】求对称饱和线性传递函数的输出对下列输入矢量的导数。

```
N=[-3;-1;0;0.5;2];
A=satlins(N);
dA_dN=dsatlins(N,A)
dA_dN =
     0
     0
     1
     1
     0
```

9．dtansig

应用：tansig 传递函数的导函数。

格式：dA_dN=dtansig (N, A)

解析：该函数用于计算 tansig 函数的输出 A 相对于输入 N 的导数。

【例 3-70】以下代码可绘制一个双曲正切 S 型传递函数的导函数。

```
N=-10:0.1:10;
A=tansig(N);
dA_dN=dtansig(N,A);
plot(N,dA_dN)
grid
```

运行结果如图 3-31 所示。

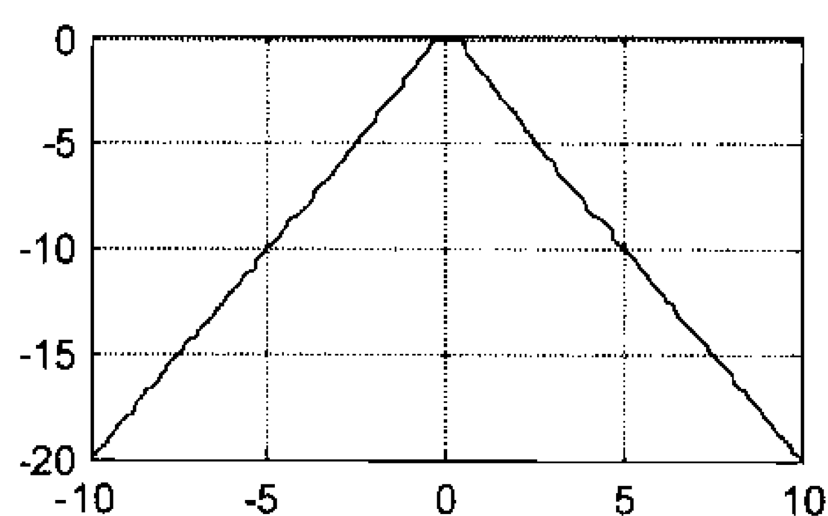


图 3-31　dtansig 的运行结果

10．dtribas

应用：三角基传递函数的导函数。

格式：dA_dN=dtribas(N, A)

解析：该函数用于计算三角基函数的输出 A 相对于输入 N 的导数。

【例 3-71】求 triabas 传递函数的输出对下列输入矢量的导数。

```
N=[-2;0.5;0.8;2];
A=tribas(N);
dA_dN=dtribas(N,A)
dA_dN =
     0
    -1
    -1
     0
```

3.9 距离函数

自组织映射网络的距离函数如表 3-9 所示。

表 3-9 自组织映射网络的距离函数

函数名称	功 能	函数名称	功 能
boxdist	Box 距离函数	dist	欧氏距离加权函数
linkdist	连接距离函数	mandist	曼哈顿距离加权函数

1. boxdist

应用： Box 距离函数。

格式： d=boxdist (pos)

解析： 在给定神经网络某层的神经元的位置后，可利用该函数计算神经元之间的距离。该函数通常用于结构函数的 gridtop 的神经网络层。其参数含义如下。

pos：神经元位置的 $N\times S$ 维矩阵；

d：函数返回值，神经元距离的 $S\times S$ 维矩阵。

函数的运算原理为 $d(i,j)=\max\|\boldsymbol{P}_i-\boldsymbol{P}_j\|$ 。其中，$d(i,j)$ 表示距离矩阵中的元素；$\boldsymbol{P}_i$ 表示位置矩阵的第 i 列向量。

【例 3-72】以下 MATLAB 代码可以演示该函数的运算规则。

```
%产生一个 3×3 的矩阵
pos=rand(3,3)
d=boxdist(pos)
```

运行结果为

```
pos =
     0.9501     0.4860     0.4565
     0.2311     0.8913     0.0185
     0.6068     0.7621     0.8214
d =
          0     0.6602     0.4937
     0.6602          0     0.8728
     0.4937     0.8728          0
```

2. linkdist

应用： 该函数为连接距离函数。

格式： d=linkdist (pos)

解析： 在给定神经元的位置后，该函数可用于计算神经元之间的距离。其参数含义参数 boxdist 函数。

函数的运算原理为

$$d(i,j)=\begin{cases}0 & 如果 i=j \\ 1 & 如果 \mathrm{sum}\left(\left(P_i-P_j\right)^2\right)^{\frac{1}{2}}\leqslant 1 \\ 2 & 如果存在 k,使得 d(i,k)=d(k,j)=1 \\ 3 & 如果存在 k_1,k_2,使得 d(i,k_1)=d(k_1,k_2)=d(k_1,j)=1 \\ N & 如果存在 k_1,k_2,\cdots,k_N,使得 d(i,k_1)=d(k_1,k_2)=\cdots=d(k_N,j)=1 \\ S & 其他\end{cases}$$

【例 3-73】下面用一组代码演示该函数的运算原理。

```
%产生一个 3×3 的矩阵
pos=rand(3,3);
d=linkdist(pos)
```

运行结果为

```
d =
     0     1     1
     1     0     1
     1     1     0
```

3．dist

应用：欧氏距离加权函数。

格式：Z=dist (W, P)

df=dist ('deriv')

d=dist (pos)

解析：通过对输入进行加权得到加权后的输入。其参数含义如下。

W：$S \times R$ 维的权值矩阵；

P：Q 组输入（列）向量的 $R \times Q$ 维矩阵；

Z：$S \times Q$ 维的距离矩阵；

df=dist ('deriv')：返回值为空，因为该函数不存在导函数。

其他参数含义请参见 boxdist 函数。

函数的运算规则为 $\boldsymbol{d}=\mathrm{sqrt}\left(\mathrm{sum}(\boldsymbol{x}-\boldsymbol{y})^2\right)$，其中 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 分别为列向量。

【例 3-74】下面用一组代码演示该函数的运行机理。

```
W=rand(3,3);
P=rand(3,1);
pos=rand(3,3);
Z=dist(W,P)
d=dist(pos)
```

运行结果为

```
Z =
    0.7306
    0.8739
    0.7668
d =
         0    0.2340    0.7917
```

```
0.2340          0    0.6875
0.7917     0.6875         0
```

4．mandist

应用：为 Manhattan 距离权函数。

格式：Z=mandist (W, P)

df=mandist ('deriv')

d=mandist (pos)

解析：各参数含义参见 dist。

函数的运算原理为 $\boldsymbol{d}=\text{sum}\left(\text{abs}\left(\boldsymbol{X}-\boldsymbol{Y}\right)\right)$，其中 $\boldsymbol{X}$ 和 $\boldsymbol{Y}$ 为两个向量。

【例 3-75】利用下面一组代码演示 mandist 函数。

```
pos=rand(3,3);
d=mandist(pos)
```

运行结果为

```
d =
          0    0.5246    0.4848
     0.5246         0    0.2317
     0.4848    0.2317         0
```

3.10　权值函数及其导函数

加权函数如表 3-10 所示，这类函数确定了神经网络每层的输入矢量和神经元权值矩阵通过何种计算方式得到神经元的传递函数的加权输入量。

表 3-10　加权函数及其导函数

函数名称	功　　能	函数名称	功　　能
dotprod	内积加权函数	ddotprod	内积加权函数的导函数
negdist	负欧氏距离加权函数	—	—
normprod	规则化内积加权函数	—	—

3.10.1　权值函数

1．dotprod

应用：内积加权函数。

格式：Z=dotprod (W, P)

df=dotprod ('deriv')

解析：它求权值与输入之间的点积作为的加权输入。其参数含义如下。

W：$S\times R$ 维的权值矩阵；

P：Q 组 R 维的输入向量；

Z：Q 组 R 维的 W 与 P 的点积；

df=dotprod ('deriv')：返回函数的导数。

【例 3-76】建立两个矩阵（或向量），演示函数 dotprod 的功能。

```
%建立一个 4×3 的矩阵，所有元素都小于 1
W=rand(4,3)
%建立一个 3×1 的矩阵，所有元素都小于 1
P=rand(3,1);
Z=dotprod(W,P);
P'
Z'
```

输出结果为

```
W =
      0.2259     0.6405     0.6808
      0.5798     0.2091     0.4611
      0.7604     0.3798     0.5678
      0.5298     0.7833     0.7942
ans =
      0.0592     0.6029     0.0503
ans =
      0.4338     0.1835     0.3025     0.5435
```

2．negdist

应用：负欧氏距离加权函数。

格式：Z=negdist (W, P)

df=negdist ('deriv')

解析：negdist 函数用于计算两矢量间的负欧氏距离，其运算原理为 $\boldsymbol{d}=-\text{sum}\left((\boldsymbol{x}-\boldsymbol{y})^2\right)^{\frac{1}{2}}$。其中，$\boldsymbol{d}$ 为矢量 $\boldsymbol{x}$ 与 $\boldsymbol{y}$ 之间的负欧氏距离。

函数调用时各参数参见 dotprod 函数。negdist 没有导函数。

【例 3-77】设定一个权值矩阵 $\boldsymbol{W}$ 和输入矢量 $\boldsymbol{P}$，计算其负欧氏距离加权输入。

```
W=[0.9 0.5 0.7;0.1 0.6 0.7];
P=[0.3 0 1]';
Z=negdist(W,P)
```

输出为

```
Z =
    -0.8367
    -0.7000
```

3．normprod

应用：规则化内积加权函数。

格式：Z=normprod (W, P)

df=normprod ('deriv')

解析：normprod 函数用于计算两矢量之间的规则化内积，其计算原理为 $\boldsymbol{Z}=\boldsymbol{W}\times\boldsymbol{P}/\text{sum}(\boldsymbol{P})$。其中，$\boldsymbol{W}$ 为权值矢量，$\boldsymbol{P}$ 为输入矢量，$\boldsymbol{Z}$ 为规则化内积。

函数参数的含义参见 dotprod 函数。normprod 函数没有导函数。

【例 3-78】设定权值矩阵 $\boldsymbol{W}$ 和输入矢量 $\boldsymbol{P}$，计算其规则化内积的加权输入。

```
W=[0.9 0.5 0.7;0.1 0.6 0.7];
P=[0.3 0 0.8]';
```

```
Z=normprod(W,P)
```

输出结果为

```
Z =
    0.7545
    0.5364
```

3.10.2　权值函数的导函数

ddotprod

应用： 内积加权函数的导函数。

格式： dZ_dP=ddotprod('p', W, P, Z)

dZ_dW=ddotprod('w', W, P, Z)

解析： ddotprod 函数用于计算内积函数的输出对输入矢量的导数。

在函数的调用形式 dZ_dP=ddotprod('p', W, P, Z)中，W 为权值矩阵，P 为该层网络的输入矢量，Z 为内积矩阵，函数返回内积函数输出对输入矢量的导数。函数的调用形式 dZ_dW=ddotprod('w', W, P, Z)返回内积函数的输出量对权值矩阵的导数。

【例 3-79】设定权值矩阵 ***W*** 和输入矢量 ***P***，计算内积加权输入及其导数。

```
W=[0.9 0.5 0.7;0.1 0.6 0.7];
P=[0.3 0 0.8]';
Z=dotprod(W,P)
dZ_dP=ddotprod('p',W,P,Z)
dZ_dW=ddotprod('w',W,P,Z)
```

输出结果为

```
Z =
    0.8300
    0.5900
dZ_dP =
    0.9000    0.5000    0.7000
    0.1000    0.6000    0.7000
dZ_dW =
    0.3000
         0
    0.8000
```

3.11　结构函数

结构函数如表 3-11 所示。

表 3-11　结构函数

函数名称	功　能
hextop	六角层结构函数
gridtop	网格层结构函数
randtop	随机层结构函数

1. hextop

应用： 该函数为六角层结构函数。

格式： pos=hextop (dim1, dim2,…, dimN)

解析： dim$i$ 维数为 i 层的长度；pos 由 N 个并列向量组成的 $N\times S$ 维矩阵，其中，S=dim1×dim2×…×dimN。

【例 3-80】利用该函数创建一个二维的神经网络层，共有 35 个神经元，分布在一个 7×5 的六角晶格上。其代码如下。

```
pos=hextop(7,5);
plotsom(pos)
```

运行结果如图 3-32 所示。

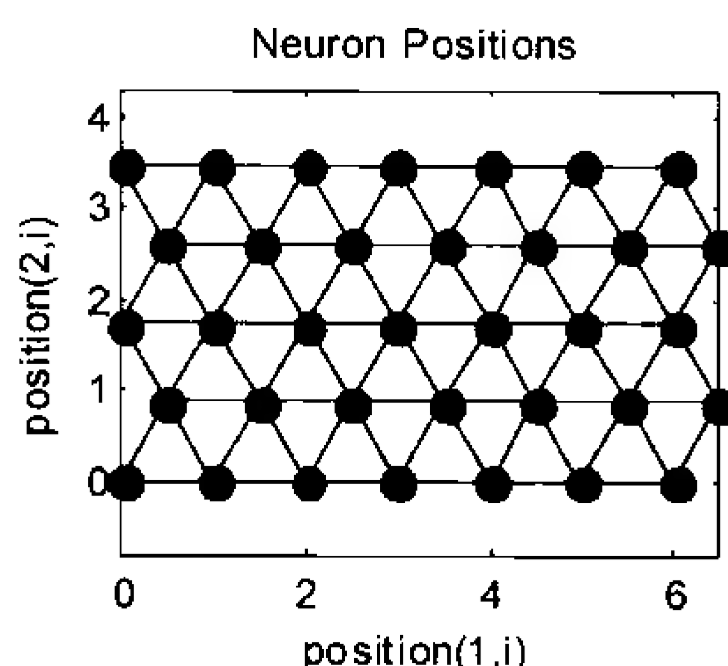


图 3-32　函数 hextop 产生的 35 个神经元的分布位置

2. gridtop

应用： 该函数为网格层结构函数。

格式： pos=gridtop (dim1, dim2,…, dimN)

解析： 各参数含义参见 hextop 函数。

【例 3-81】利用该函数创建一个二维的神经网络层，共有 35 个神经元，分布在一个 7×5 的网格上。代码如下。

```
pos=gridtop(7,5);
plotsom(pos)
```

运行结果如图 3-33 所示。

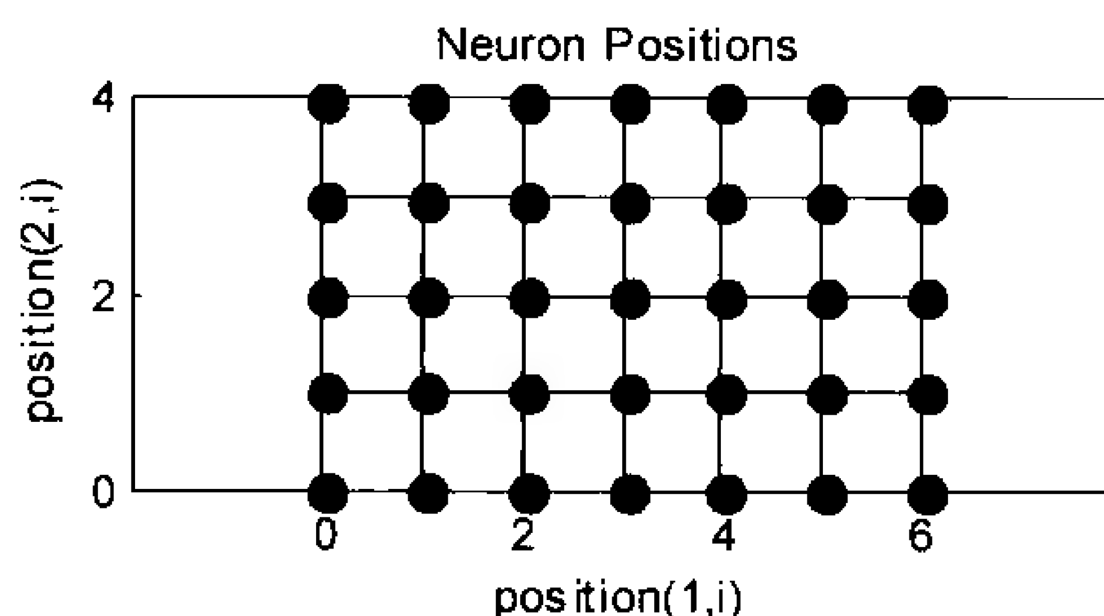


图 3-33　函数 gridtop 产生的 35 个神经元的分布位置

3. randtop

应用： 该函数为随机层结构函数。

格式： pos=randtop (dim1, dim2,…, dimN)

解析： 各参数含义参见 hextop。

【例 3-82】利用该函数创建一个二维的神经网络层，共有 35 个神经元，分布在一个 7×5

的随机网格上。代码如下。

```
pos=randtop(7,5);
plotsom(pos)
```

运行结果如图 3-34 所示。

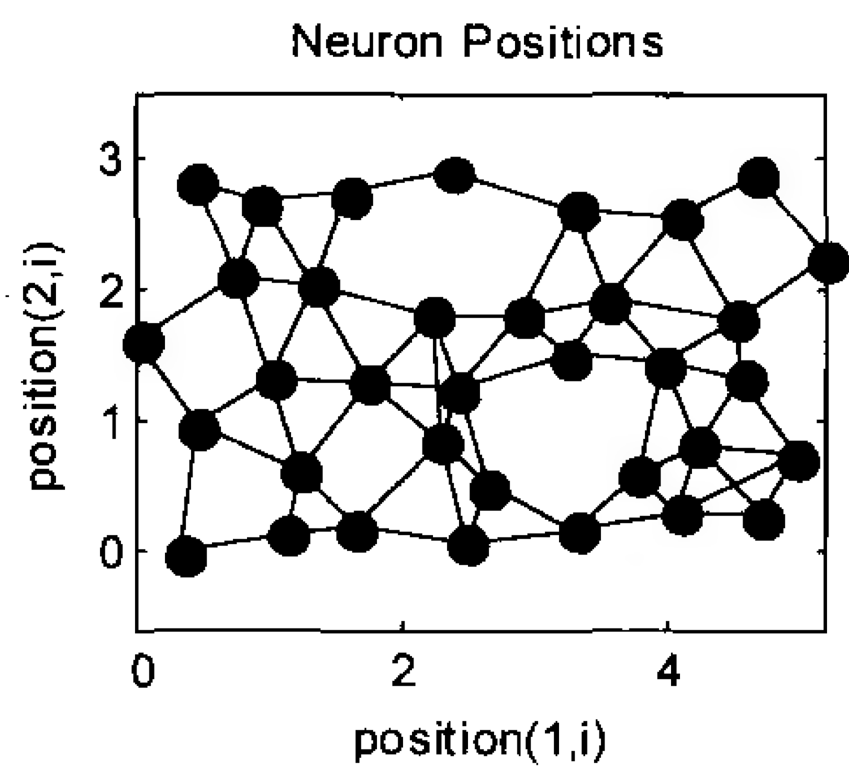


图 3-34　函数 randtop 产生的 35 个神经元的分布位置

3.12　分析函数

分析函数如表 3-12 所示。

表 3-12　分析函数

函数名称	功　能
errsurf	计算单输入神经元的误差曲面
maxlinlr	求取线性神经网络的最大学习速率

1. errsurf

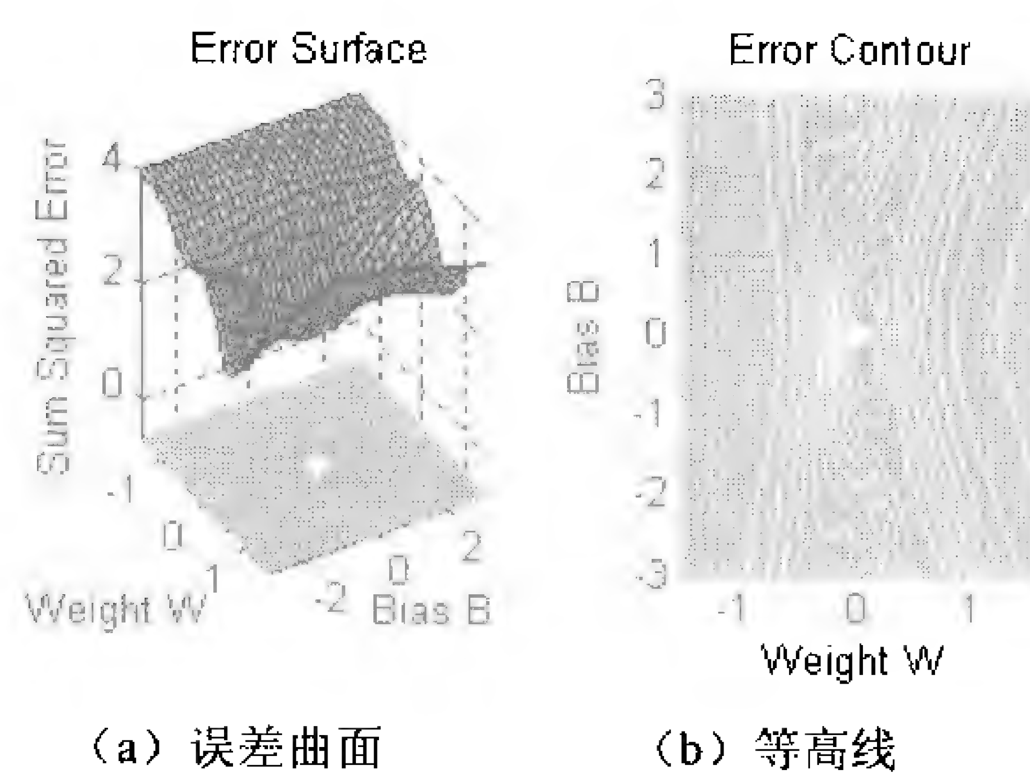
应用： 计算单输入神经元的误差曲面。

格式： E=errsurf (P, T, WV, BV, F)

解析： 该函数可用于计算单输入神经元在给定的权值范围矢量和阈值范围矢量的情况下的网络误差。函数的输入为神经元矢量 P、目标矢量 T、权值范围矢量 WV、阈值范围矢量 BV 以及神经元传递函数 F，函数返回误差曲面各点的误差 E。

【例 3-83】下面的一组代码可以分析一个 BP 网络中某个神经元的误差，并绘制出其误差曲面与轮廓线。

```
P=[-6.1 6 -4.1 -4 4 4.1 -6 6.1];
T=[0 1 0.89 0.92 0.02 0.04 0 1];
WV=-1.5:0.1:1.5;
BV=-3:0.2:3;
ES=errsurf(P,T,WV,BV,'logsig');
%绘制误差曲面和等高线，如图 3-35 所示
plotes(WV,BV,ES,[60 30])
W=0;
B=0;
E=sse(T-logsig(W*P+B));
plotep(W,B,E);
```



（a）误差曲面　　（b）等高线

图 3-35　误差曲面和等高线

2. maxlinlr

应用：求取线性神经网络的最大学习速率。

格式：lr=maxlinlr (P)

lr=maxlinlr (P, 'bias')

解析：maxlinlr 函数可以根据线性神经网络的输入矢量 P 设定网络的学习速率。当网络神经元没有阈值时，使用函数形式 lr=maxlinlr (P)来获得权值的学习速率；当网络神经元有阈值时，使用形式 lr=maxlinlr (P, 'bias')得到权值和阈值的学习速率。

【例 3-84】下面一组代码可在给定输入 ***P*** 的情况下，分“带阈值”和“不带阈值”两种情况求得该线性层所需的最大学习速率。

```
P=[1 2 -3 7;0.2 4 10 7];
lr1=maxlinlr(P,'bias')
lr2=maxlinlr(P)
```

运行结果为

```
lr1 =
    0.0057
lr2 =
    0.0058
```

3.13　转换函数

转换函数如表 3-13 所示。

表 3-13　转换函数

函数名称	功　　能
sp2narx	转换串联输入为平行输入
ind2vec	将数据索引转换为向量组
vec2ind	ind2vec 的逆函数
cell2mat	将矩阵构成的单元数组组合成矩阵
mat2cell	将矩阵分裂为由子矩阵构成的单元数组
combvec	建立矢量所有组合的矩阵
con2seq	将并行矢量转换为串行矢量
concur	复制阈值矢量并构成矩阵

续表

函数名称	功　能
minmax	求矩阵每行的范围
normr	正则化矩阵的行
normc	正则化矩阵的列
pnormc	伪正则化矩阵的列
quant	将实数量化为某一数值的整数倍
sumsqr	求矩阵的平方和
seq2con	将串行矢量转换为并行矢量

1．sp2narx

应用：转换串联输入为平行输入。

格式：net=sp2narx (net)

解析：其中 net=sp2narx 用于把一个串联输入非线性回归神经网络转换为平行输入；net 为原来的串联输入非线性回归神经网络。

2．ind2vec

应用：用于将数据索引转换为向量组。

格式：vec=ind2vec (ind)

解析：ind 为数据索引列向量；vec 为函数返回值，一个稀疏矩阵，每行只有一个 1，矩阵的行数等于数据索引的个数，列数等于数据索引中的最大值。

【例 3-85】可通过下面的一组代码演示 ind2vec 的计算原理。

```
ind=[3 5 7 9];
vec=ind2vec(ind)
```

上述代码的第一行定义了一个数据索引列向量，第二行将其转换为向量组。运行结果为

```
vec =
    (3,1)        1
    (5,2)        1
    (7,3)        1
    (9,4)        1
```

可以看出，结果应该是一个 4×3 的矩阵，其中（x,y）是与数据索引列向量相对应的。x 为数据索引值，y 表示 x 在数据索引中的位置，由此组成了输出矩阵。矩阵中没有填满的部分需要用 0 补齐。

3．vec2ind

该函数用于将向量组转换为数据索引，与 ind2vec 是互逆的。

4．cell2mat

应用：将矩阵构成的单元数组组合成矩阵。

格式：m=cell2mat (c)

解析：cell2mat 函数可以把矩阵构成的单元数组重组为矩阵。函数的输入 c 是一个单元数组，函数返回由 c 中各元素构成的矩阵。

【例 3-86】

```
c={[1] [3];[4;8] [5;7]};
```

```
m=cell2mat(c)
```

输出为

```
m =
        1      3
        4      5
        8      7
```

5. combvec

应用： 建立矢量所有组合的矩阵。

格式： a=combvec (a1, a2,⋯, aN)

解析： combvec 函数将建立输入矩阵 a1 到 aN 中各列矢量的所有组合，并返回由这些组合构成的矩阵。

【例 3-87】

```
a1=[1 3;5 7];
a2=[2;4];
a3=combvec(a1,a2)
```

输出为

```
a3 =
        1      3
        5      7
        2      2
        4      4
```

6. con2seq

应用： 将并行矢量转换为串行矢量。

格式： s=con2seq (b)

s=con2seq (b,TS)

解析： 在神经网络工具箱中，矩阵用于存储神经网络的并行输入矢量，单元数组用于存储网络的串行输入矢量。con2seq 函数可以实现网络并行输入矢量到串行输入矢量的转换。

在函数的调用 s=con2seq (b)中，函数输入矩阵为 b，返回单元数组为 s，s 中的每一个元素为矩阵 b 的列矢量。在函数的调用 s=con2seq (b,TS)中，函数输入 b 为 $N\times 1$ 维单元数组，b 中的每一个元素是由 $M\times$TS 列并行输入矢量构成的矩阵，TS 为网络的工具步数；函数返回 $N\times$TS 维单元数组 s，s 中的每一元素是由 M 列并行输入矢量构成的矩阵。

【例 3-88】

```
P1=[1 3 5];
P2=con2seq(P1)
```

输出为

```
P2 =
    [1]    [3]    [5]
```

7. concur

应用： 复制阈值矢量并构成矩阵。

格式： concur (B, Q)

解析：concur 函数的输入 B 为阈值矢量，Q 为复数次数，函数返回的矩阵是由 Q 个原阈值矢量构成的矩阵。

【例 3-89】

```
B=[2;4;6];
concur(B,3)
```

输出为

```
ans =
     2     2     2
     4     4     4
     6     6     6
```

8．mat2cell

应用：将矩阵分裂由子矩阵构成的单元数组。

格式：cell=mat2cell (M, R, C)

解析　mat2cell 函数对输入矩阵 M 进行分裂，分裂后各子矩阵的行数由矢量 R 决定，列数由矢量 C 决定。函数返回由各子矩阵构成的单元数组 cell。

【例 3-90】

```
M=[1 4 7;2 5 8;3 6 9];
C=mat2cell(M,[2 1],[1 2])
```

输出为

```
C =
    [2x1 double]    [2x2 double]
    [          3]    [1x2 double]
```

9．minmax

应用：求矩阵每行的范围。

格式：PR=minmax (P)

解析：minmax 函数求取输入矩阵 P 中每一行矢量的取值范围，并返回各行中最小值和最大值构成的范围矩阵 PR。

【例 3-91】

```
P=[-1 -5 6 1;7 2 -1 4];
PR=minmax(P)
```

输出为

```
PR =
    -5     6
    -1     7
```

10．normr

应用：正则化矩阵的行。

格式：N=normr (M)

解析：normr 函数对其输入矩阵 M 进行正则化，函数返回的正则化矩阵 N 与原始矩阵维数相同，矩阵每一行元素的平方和变为 1，但各元素之间的比例不变。

【例 3-92】

```
M=[3 5;7 9;2 6];
N=normr(M)
```

输出为

```
N =
    0.5145    0.8575
    0.6139    0.7894
    0.3162    0.9487
```

11．normc

应用：正则化矩阵的列。

格式：N=normc (M)

解析：normc 函数对其输入矩阵 M 进行正则化，函数返回的正则化矩阵 N 与原始矩阵维数相同，矩阵每一列元素的平方和变为 1，但各元素之间的比例不变。

【例 3-93】

```
M=[3 5;7 9;2 6];
N=normc(M)
```

输出为

```
N =
    0.3810    0.4196
    0.8890    0.7553
    0.2540    0.5035
```

12．pnormc

应用：伪正则化矩阵的列。

格式：N=pnormc (M, R)

解析：pnormr 函数通过对输入矩阵 M 的每一列添加一个元素得到伪正则化矩阵 N，N 中每一列元素的平方和等于输入参数 R 的平方。

【例 3-94】

```
M=[3 5;7 9;2 8];
N=pnormc(M,6)
```

输出为

```
N =
    3.0000              5.0000
    7.0000              9.0000
    2.0000              8.0000
    0 + 5.0990i         0 +11.5758i
```

13．quant

应用：将实数量化为某一数值的整数倍。

格式：quant (X,Q)

解析：quant 函数将矩阵 X 中的每个元素离散化为量化因子 Q 的最近整数倍。

【例 3-95】

```
X=[1.6372 5.3578 -1.7892];
Y=quant(X,0.2)
```

输出为

```
Y =
     1.6000     5.4000    -1.8000
```

14．seq2con

应用：将串行矢量转换为并行矢量。

格式：b=seq2con (s)

解析：在神经网络工具箱中，矩阵用于存储神经网络的并行输入矢量，单元数组用于存储网络的串行输入矢量。seq2con 函数可以实现网络串行输入矢量到并行输入矢量的转换。

函数输入 s 为 $N\times$TS 维单元数组，s 中的每一个元素是由 M 列矢量构成的矩阵。函数返回 $N\times1$ 维的单元数组 b，b 中的每一个元素是由 $M\times$TS 列并行输入矢量构成的矩阵。

【例 3-96】

```
p1={1 5 9};
p2=seq2con(p1)
```

输出为

```
p2 =
     [1×3 double]
```

由上述结果可知，单元数组 p2 中的唯一元素是一个 1×3 维矩阵。

```
p2{1}
ans =
       1       5       9
```

15．sumsqr

应用：求矩阵的平方和。

格式：sumsqr (m)

解析：sumsqr 函数用来求取函数输入矩阵 m 中各元素的平方和。

【例 3-97】

```
s=sumsqr([3 7;2 8])
s =
    126
```

3.14　绘图函数

绘图函数如表 3-14 所示。

表 3-14　绘图函数

函数名称	功　能	函数名称	功　能
hintonw	绘制权值矩阵的 Hinton 图	plotpc	在感知器输入矢量图上绘制分界线
hintonwb	绘制权值矩阵和阈值矢量的 Hinton 图	plotperf	绘制网络训练过程中的性能变化曲线

续表

函数名称	功　能	函数名称	功　能
plotbr	绘制网络采用 Bayesian 正则化算法训练时的性能变化曲线	plotsom	绘制自组织映射网络图
		plotv	绘制起自原点的矢量
plotes	绘制单输入神经元的误差曲面	plotvec	用不同颜色绘制矢量
plotpv	根据目标矢量绘制感知器的输入矢量	plotep	在单输入神经元的误差曲面上绘制权值、阈值和相应误差点的位置

1．plotpc

应用：分界线绘制函数。

格式：plotpc (W, b)

plotpc (W, b, h)

解析：该函数用于在感知器矢量图中绘制分界线，其参数含义如下。

W：$S\times R$ 维的加权矩阵（R 必须$\leqslant 3$）；

b：$S\times 1$ 维的阈值向量；

h：最后画线的控制权；

plotpc (W, b)：返回的是对所绘制分界线的控制权；

plotpc (W, b, h)：用于在绘制的新线之间检查最新绘制的分界线。

2．plotpv

应用：输入/目标向量绘制函数。

格式：plotpv (p, t)

plotpv (p, t, v)

解析：该函数用于绘制感知器的输入向量和目标向量。其参数含义如下。

p：Q 组 R 维的输入向量；

t：Q 组 S 维的双目标向量；

v=[x_min x_max y_min y_max]：图形的最大值，绘制工作必须位于 v 所限定的范围内；

plotpv (p, t)：以 t 为标尺，绘制 p 的列向量；

plotpv (p, t, v)：在 v 的范围中绘制 p 的列向量。

【例 3-98】本例主要目的在于演示函数 plotpc 和 plotpv 的应用。

假定已给出了某感知器的输入变量 $\boldsymbol{p}$ 和目标变量 $\boldsymbol{t}$，绘制其曲线；然后给定感知器的权值和阈值，绘制其分界线。

MATLAB 代码如下。

```
p=[0 0 1 1;0 1 0 1];
t=[0 0 0 1];
%绘制输入向量和目标向量
plotpv(p,t)
net=newp(minmax(p),1); %创建一个感知器网络
%设定权值
net.iw{1,1}=[-1.2 -0.5];
net.b{1}=1;   %设定阈值
plotpc(net.iw{1,1},net.b{1})
```

运行结果如图 3-36 所示。

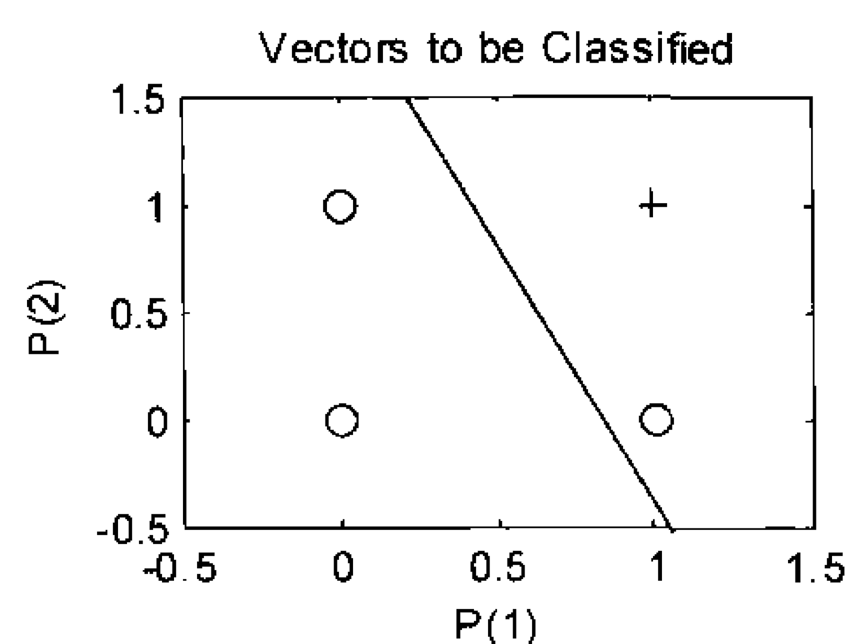


图 3-36　函数 plotpc 和 plotpv 演示运行结果

3．hintonw

应用：绘制权值矩阵的 Hinton 图。

格式：hintonw (W, maxw, minw)

解析：hintonw 函数以方块图的形式表示权值矩阵，图中各方块的面积正比于权值矩阵中相应元素的大小，方块的颜色表示该元素的符号。其中，深色表示负权值，浅色表示正权值。该函数的输入为权值矩阵 W、最大权值 maxw 和最小权值 minw。

【例 3-99】根据下面的权值矩阵绘制 Hinton 图。

```
W=[0.9631 -0.2345 0.0235 -0.0123;
   -0.3245 0.8963 -0.6348 -0.3517;
   0.5681 0.3964 -0.6321 0.8527]
hintonw(W)
```

运行结果如图 3-37 所示。

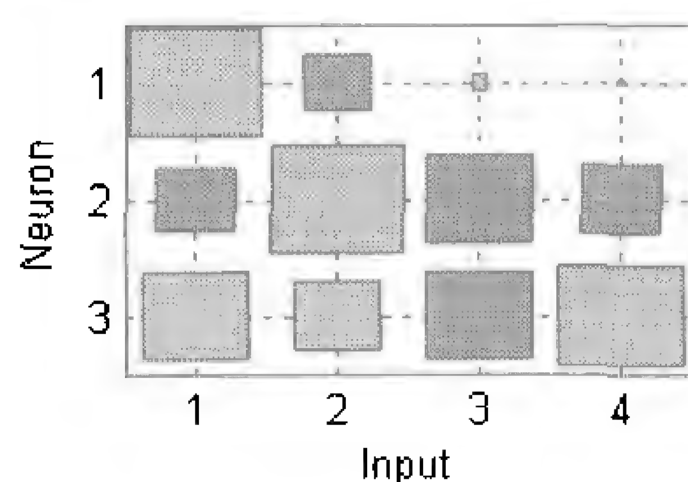


图 3-37　权值的 Hinton 图

4．hintonwb

应用：绘制权值矩阵和阈值矢量的 Hinton 图。

格式：hintonwb (W, b, maxw, minw)

解析：hintonwb 函数以方块图的形式表示权值矩阵和阈值矢量，图中各方块的面积正比于权值矩阵或阈值矢量中相应元素的大小，方块的颜色表示该元素的符号，其中，深色表示负值，浅色表示正值。该函数的输入为权值矩阵 W、阈值矢量 b、最大权值 maxw 和最小权值 minw。

【例 3-100】根据下面的权值矩阵和阈值矢量绘制 Hinton 图。

```
W=[0.9631 -0.2345 0.0235 -0.0123;
   -0.3245 0.8963 -0.6348 -0.3517;
   0.5681 0.3964 -0.6321 0.8527];
b=[0.4536;-0.6321;0.1235];
hintonwb(W,b)
```

运行结果如图 3-38 所示。

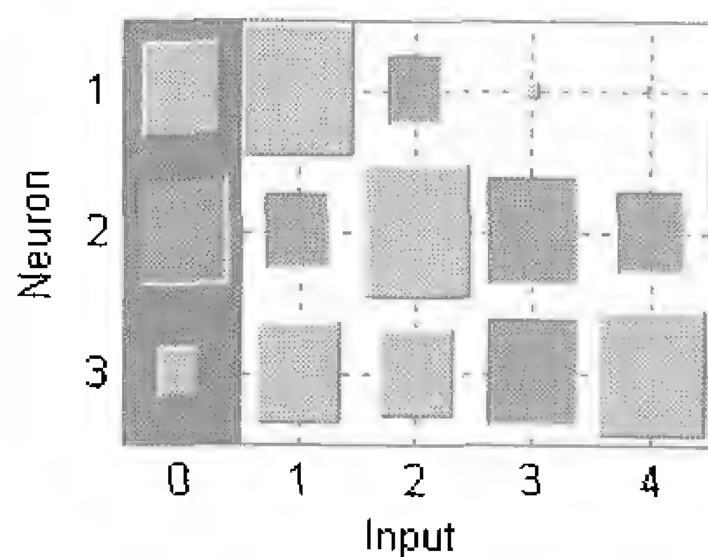


图 3-38 权值和阈值的 Hinton 图

5．plotbr

应用：绘制网络采用 Bayesian 正则化算法训练时的性能变化曲线。

格式：plotbr (TR, name, epoch)

解析：当网络的训练函数为 trainbr 时，可以利用 plotbr 函数绘制训练过程中网络性能的变化曲线，函数绘制的曲线包括网络输出的方差和、各权值参数的平方和，以及网络中有效参数的个数。函数的输入 TR 为训练函数产生的训练记录；name 为训练函数名称，默认值为空字符串；epoch 为要显示的训练次数，其默认值为训练记录的长度。

【例 3-101】利用下面一组代码演示 plotbr 函数记录训练过程曲线。

MATLAB 代码如下。

```
p=[-1:0.1:1];
t=sin(2*3.14159*p)+0.15*randn(size(p));
net=newff([-1 1],[20,1],{'tansig','purelin'},'trainbr');
[net,tr]=train(net,p,t);
plotbr(tr);
```

运行结果如图 3-39 所示。

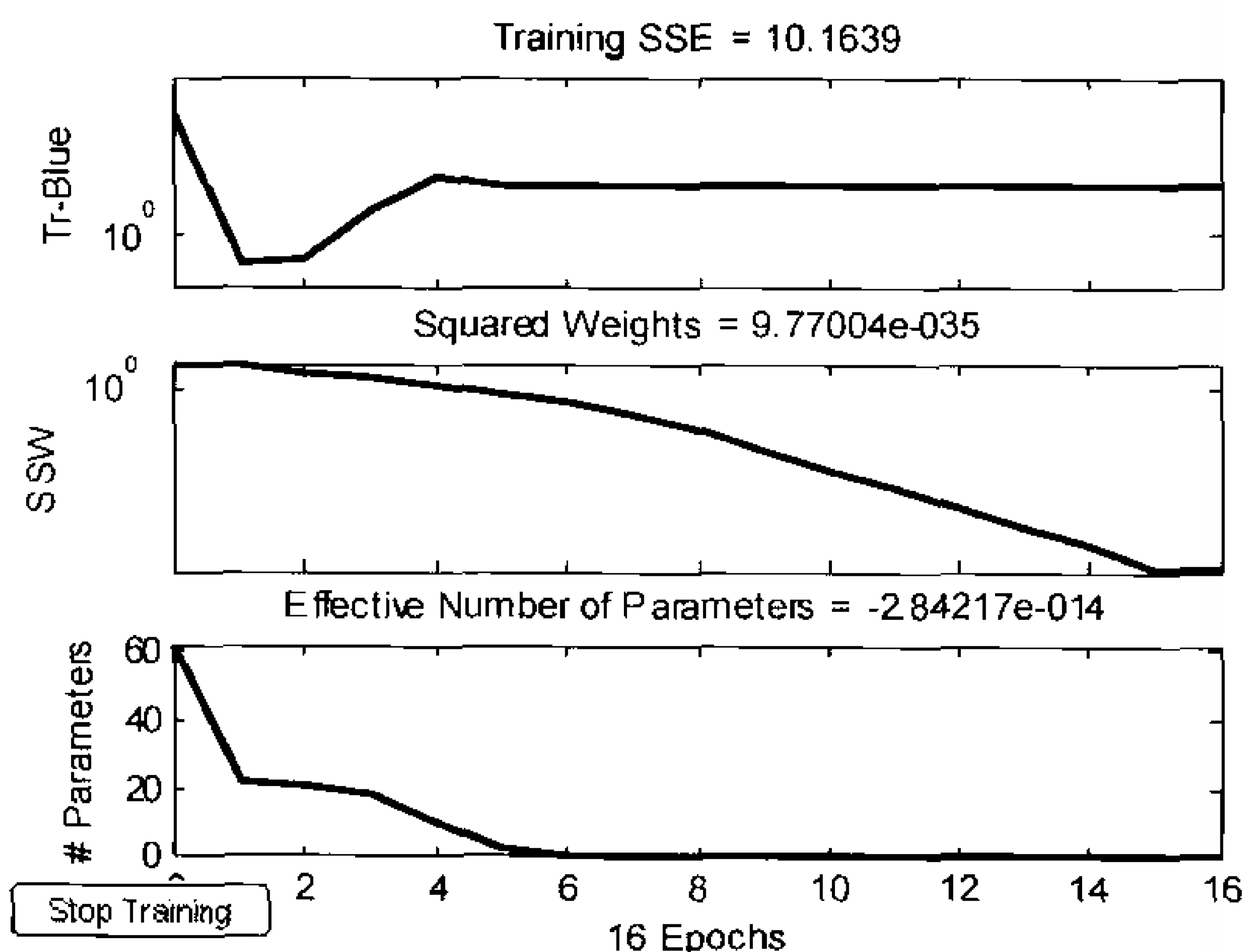


图 3-39 Bayesian 正则化训练时的网络性能变化曲线

6．plotes

应用：用于绘制一个单输入神经元的误差曲面。

格式：plotes (wv, bv, es, v)

解析：plotes 函数可用来绘制单输入神经元的误差曲面和等高线。wv 为神经元可能的权值范围矢量；bv 为神经元可能的阈值范围矢量；es 是由 errsurf 函数计算出的误差矩阵，该矩阵给出了在权值范围矢量和阈值范围矢量各组合点上的神经元输出误差；v 给出了三维图形的视点，默认值为[-37.5, 30]。

示例参看例 3-83。

7．plotep

应用：在单输入神经元的误差曲面上绘制指定权值、阈值及相应误差点的位置。

格式：H=plotep (W, B, E)

H=plotep (W, B, E, H)

解析：plotep 函数可以在 plotes 函数所绘的图形上标识出指定权值、阈值和相应误差点的位置。在函数的输入中，W 为权值，B 为阈值，E 为误差，plotep 函数将在等高线图上绘出对应于 W 和 B 的点，在误差曲面上绘出对应于 E 的点，同时函数返回的句柄 H 保存了本次函数所绘各点的信息。调用 H=plotep (W, B, E, H)利用上次调用函数时返回的句柄 H，在绘制新点前删除旧点。

示例参看例 3-83。

8．plotperf

应用：绘制网络训练过程中的性能变化曲线。

格式：plotperf (TR, goal, name, epoch)

解析：plotperf 函数用于绘制网络在训练时的性能变化曲线，如果训练中有验证步骤和测试步骤，本函数还将绘制验证性能和测试性能的变化曲线。函数的输入 TR 为训练记录；goal 为网络性能目标，默认值为 NaN；name 为训练函数的名称；epoch 为要显示的训练次数，其默认值为训练记录的长度。

【例 3-102】有一个输入矢量 ***P*** 和目标矢量 ***T***，分别有 8 个向量。由此导出一组确认样本如下。

```
P=1:8;
T=sin(P);
VV.P=P;
VV.T=T+rand(1,8)*0.1;
```

创建一个 BP 网络，并进行训练，找出 ***P*** 和 ***T*** 之间的非线性关系，并利用确认样本对网络进行检验。

```
net=newff(minmax(P),[4 1],{'tansig','tansig'});
[net,tr]=train(net,P,T,[],[],VV);
plotperf(tr);
```

网络的训练误差曲线如图 3-40 所示，训练结果为

```
TRAINLM, Epoch 0/100, MSE 1.32542/0, Gradient 4.61101/1e-010
TRAINLM, Epoch 14/100, MSE 0.374589/0, Gradient 0.00939335/1e-010
TRAINLM, Validation stop.
```

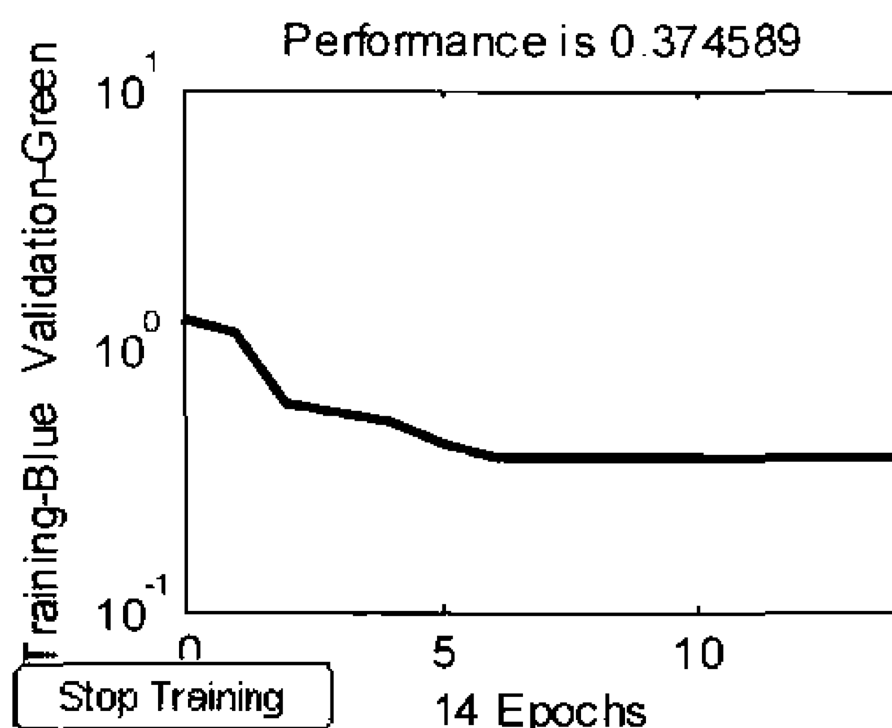


图 3-40　网络的训练误差曲线

9. plotsom

应用：绘制自组织映射网络图。

格式：plotsom (pos)

plotsom (W, D, nd)

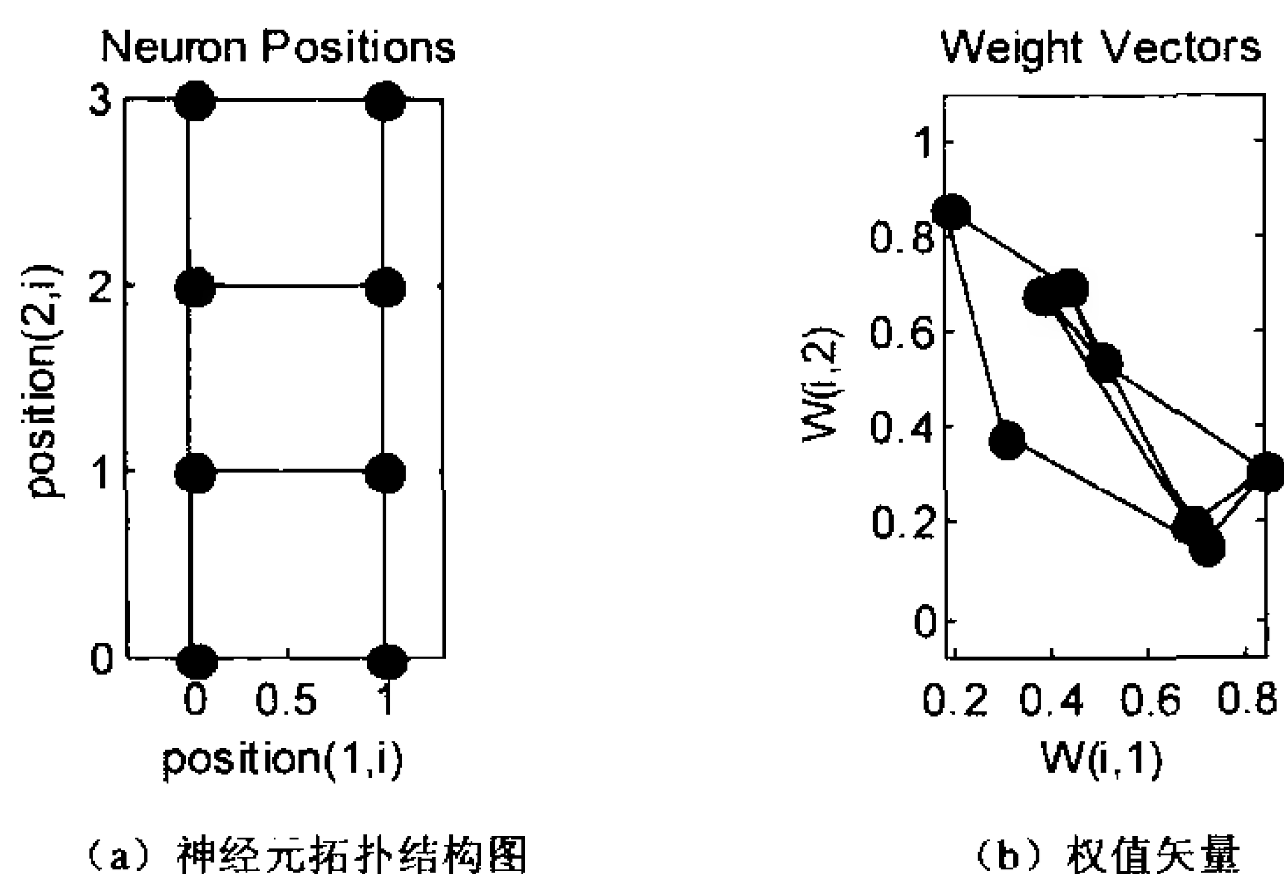
解析：plotsom 函数用于绘制自组织映射网络图。在函数调用 plotsom (pos)中，pos 是网络中各神经元在物理空间分布的位置坐标矩阵；函数返回神经元物理分布的拓扑图，图中每两个间距小于 1 的神经元以直线连接。在函数调用 plotsom (W, D, nd)中，W 为神经元权值矩阵；D 为根据神经元位置坐标计算出来的间距矩阵；nd 为领域半径，默认值为 1；函数返回神经元权值的分布图，图中每两个间距小于 nd 的神经元以直线连接。

【例 3-103】构造一个 2 输入 8 神经元的自组织映射网络层，网络的拓扑结构为长方形，神经元间距采用 linkdist 计算方法并随机产生权值矩阵 ***W***。

MATLAB 代码如下。

```
pos=gridtop(2,4);
W=rand(8,2);
%绘制神经元拓扑结构图和权值矢量;
subplot(1,2,1)
plotsom(pos);
D=linkdist(pos);
subplot(1,2,2)
plotsom(W,D)
```

运行结果如图 3-41 所示。



（a）神经元拓扑结构图　（b）权值矢量

图 3-41　自组织映射网络中的神经元拓扑结构图和权值矢量

10. plotv

应用：绘制起自原点的矢量。

格式：plotv (M, t)

解析：plotv 函数用于绘制矢量图，函数的输入 M 为矢量矩阵；t 指定绘图时矢量线的形状，默认值为'-'；函数将绘制 M 中的每一个列矢量。

【例 3-104】绘制下面的矢量。

```
w=[-0.4 0.7;-0.5 0.2];
plotv(w,'ro')
```

运行结果如图 3-42 所示。

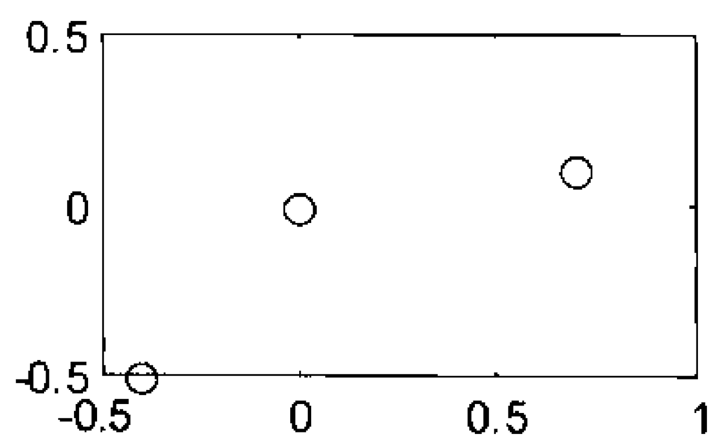


图 3-42　矢量图

11．plotvec

应用：用不同颜色绘制矢量。

格式：plotvec (X, C, m)

解析：plotvec 函数利用不同颜色绘制矢量。函数的输入 X 为矢量矩阵；C 是表示颜色坐标的行矢量；m 指定绘图时矢量的标识符号。函数将绘制 X 中的每一个列矢量，每个矢量的颜色由矢量 C 中的对应元素确定。

【例 3-105】针对一组输入向量，设计一个 LVQ 神经网络，经过训练后，能对给定数据进行模式识别。

MATLAB 代码如下。

```
P=[-6 -4 -2 0 0 0 1 2 4 6;0 2 -2 1 2 -2 1 2 -2 0];
C=[1 1 1 2 2 2 2 1 1 1];
T=ind2vec(C);
plotvec(P,C,'+b');
axis([-6 6 -3 3]);
net=newlvq(minmax(P),5,[0.6 0.4]);
net.trainParam.epochs=85;
net=train(net,P,T);
p=[0 1;0.2 0];
y=sim(net,p);
vc=vec2ind(y)
```

运行结果如图 3-43 和图 3-44 所示。

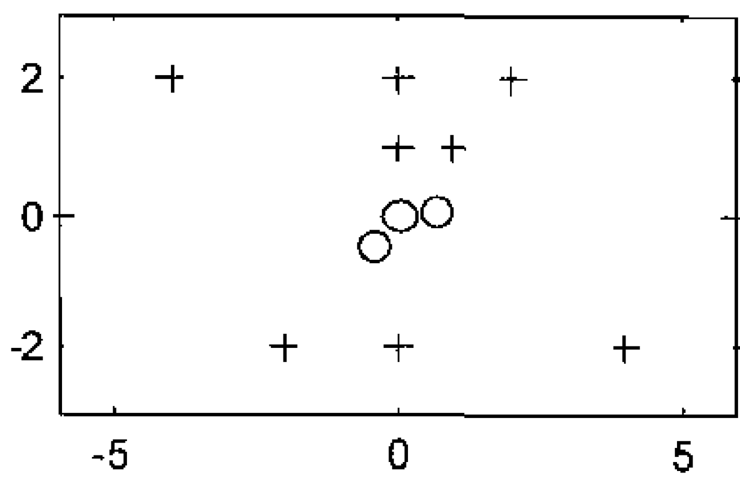


图 3-43　输入样本数据分布图

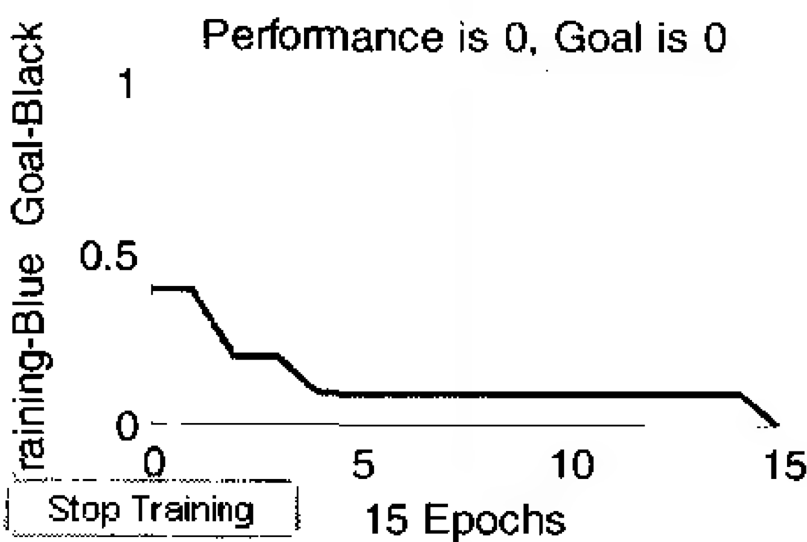


图 3-44　训练误差曲线

在命令窗口中得到下面的结果。

```
%用 TRAINR 作为训练函数，最大训练次数为 85 次
RAINR, Epoch 0/85
TRAINR, Epoch 15/85
TRAINR, Performance goal met.
%对给定数据，一个归于第 2 类，一个归于第 1 类
vc =
      2      1
```

3.15 数据预处理和后处理函数

数据预处理和后处理函数如表 3-15 所示。

表 3-15 数据预处理和后处理函数

函数名称	功 能
premnmx	把数据归一化到-1～1 之间
postmnmx	恢复被函数 premnmx 归一化的数据
prestd	把数据归一化为单位方差和零均值
poststd	恢复被函数 prestd 归一化的数据
postreg	利用线性回归分析对神经网络的仿真结果进行后处理
prepca	对输入数据进行主元分析
trapca	利用预先计算的主元分析矩阵对数据进行变换
trastd	利用预先计算的均值和方差对数据进行变换
tramnmx	利用预先计算的最大值和最小值对数据进行变换

1. premnmx

应用： 把数据归一化到-1～1 之间。

格式： [Pn, minp, maxp, Tn, mint, maxt]=premnmx (P, T)

[Pn, minp, maxp]=premnmx (P)

解析： premnmx 函数用于对网络的输入数据或目标数据进行归一化，归一化后的数据将分布在[-1，1]区间内。归一化表达式为

$$Pn=2*(P-minp)/(maxp-minp)-1$$

$$Tn=2*(T-mint)/(maxt-mint)-1$$

其中，P 为原始输入数据，maxp 和 minp 分别是 P 中的最大值和最小值，Pn 为归一化后的输入数据。T 是原始目标数据，maxt 和 mint 分别是 T 中的最大值和最小值，Tn 是归一化后的目标数据。

函数调用[Pn, minp, maxp, Tn, mint, maxt]=premnmx (P, T)可以把网络输入数据 P 和目标数据 T 归一化为 Pn 和 Tn，同时返回 P 中的最小值 minp 和最大值 maxp，以及 T 中的最小值 mint 和最大值 maxt。也可以调用[Pn, minp, maxp]=premnmx (P)形式只对输入数据 P 进行归一化。

2. postmnmx

应用： 恢复被函数 premnmx 归一化的数据。

格式： [P, T]=postmnmx (Pn, minp, maxp, Tn, mint, maxt)

[P]= postmnmx (Pn, minp, maxp)

解析： 该函数用于恢复被函数 premnmx 归一化的数据，变换表达为

$$P=0.5*(Pn+1)*(maxp-minp)+minp$$

其中，P 为原始数据，maxp 和 minp 分别是 P 中的最大值和最小值。Pn 为归一化后的数据。同理

$$T=0.5*(Tn+1)*(maxt-mint)+mint$$

函数调用[P, T]=postmnmx (Pn, minp, maxp, Tn, mint, maxt)可以在已知 P 中的最小值 minp 和最大值 maxp 及 T 中的最小值 mint 和最大值 maxt 的前提下，把归一化的网络输入数据 Pn 和目标数据 Tn 恢复为原始数据 P 和 T。也可以利用函数[P]= postmnmx (Pn, minp, maxp)形式只恢复归一化输入数据 P。

【例 3-106】把下列数据归一化到[-1，1]区间，并恢复其原始数据。

```
P=[-9 -6 -4.5 -1.5 0];
T=[0 8.8 -11 -10 2];
%首先对数据进行归一化
[Pn,minp,maxp,Tn,mint,maxt]=premnmx(P,T)
Pn =
   -1.0000    -0.3333         0    0.6667    1.0000
minp =
    -9
maxp =
     0
Tn =
    0.1111    1.0000   -1.0000   -0.8990    0.3131
mint =
   -11
maxt =
    8.8000
%恢复原始数据
[P,T]=postmnmx(Pn,minp,maxp,Tn,mint,maxt)
P =
   -9.0000   -6.0000   -4.5000   -1.5000         0
T =
         0    8.8000  -11.0000  -10.0000    2.0000
```

3．prestd

应用： 把数据归一化为单位方差和零均值。

格式： [Pn, meanp, stdp, Tn, meant, stdt]=prestd (P, T)

[Pn, meanp, stdp]=prestd (P)

解析： prestd 函数用于对网络的输入数据或目标数据进行归一化，归一化后的数据将具有零均值和单位方差。归一化表达为

$$Pn=(P-meanp)/stdp$$

其中，P 和 Pn 分别为归一化前、后的输入数据，meanp 和 stdp 分别为原始数据 P 的均值和方差。同理

$$Tn=(T-meant)/stdt$$

其中，T 和 Tn 分别是归一化前、后的目标数据，meant 是原始数据 T 的均值，stdt 是其方差。

函数调用[Pn, meanp, stdp, Tn, meant, stdt]=prestd (P, T)可以把网络的输入数据 P 和目标数据 T 归一化为 Pn 和 Tn，同时返回 P 的均值 meanp 和方差 stdp 及 T 的均值 meant 和方差 stdt。也可以调用[Pn, meanp, stdp]=prestd (P)只对输入数据 P 进行归一化。

4．poststd

应用： 恢复被函数 prestd 归一化的数据。

格式：[P, T]=poststd (Pn, meanp, stdp, Tn, meant, stdt)
[P]=poststd (Pn, meanp, stdp)

解析：poststd 函数用于恢复被函数 prestd 归一化的函数，变换表达为

$$P=Pn*stdp+meanp$$

其中，P 和 Pn 分别为归一化前、后的数据，meanp 和 stdp 分别为原始数据 P 的均值和方差。同理

$$T=Tn*stdt+meant$$

函数调用[P, T]=poststd (Pn, meanp, stdp, Tn, meant, stdt)可以把归一化数据 Pn 和 Tn 恢复为原始数据 P 和 T，同时需要已知 P 的均值 meanp 和方差 stdp 及 T 的均值 meant 和方差 stdt。也可以调用[P]=poststd (Pn, meanp, stdp)只恢复归一化输入数据 P。

【例 3-107】把下列数据归一化为具有零均值和单位方差的数据序列并恢复其原始数据。

```
P=[-9 -6 -4.5 -1.5 0];
T=[0 8.8 -11 -10 2];
%首先对数据进行归一化
[Pn,meanp,stdp,Tn,meant,stdt]=prestd(P,T)
Pn =
    -1.3403   -0.5026   -0.0838    0.7539    1.1728
meanp =
    -4.2000
stdp =
     3.5812
Tn =
     0.2431    1.2919   -1.0678   -0.9486    0.4815
meant =
    -2.0400
stdt =
     8.3909
%恢复原始数据
[P,T]=poststd(Pn,meanp,stdp,Tn,meant,stdt)
P =
    -9.0000   -6.0000   -4.5000   -1.5000         0
T =
          0    8.8000  -11.0000  -10.0000    2.0000
```

5．postreg

应用：利用线性回归分析对神经网络的仿真结果进行后处理。

格式：[M, B, R]=postreg (A, T)

解析：该函数对网络的仿真输出和目标矢量进行线性回归分析，并得到目标矢量对网络输出的相关系数，从而可以作为检验网络性能的参数。

函数的输入分别为网络输出矢量 A 和目标矢量 T，函数返回线性拟合直线的斜率为 M，截距系数为 B，输出矢量 A 和目标矢量 T 之间的相关系数为 R。当 R 为 1 时，输出和目标矢量之间的相关性最好。

【例 3-108】利用级联前向网络对下列样本数据进行学习，并对网络的实际输出矢量和目标矢量进行线性回归分析。

MATLAB 代码如下。

```
P=[0.96 -0.78 -0.53 0.74 0.31];
net=newff(minmax(P),[5 1],{'tansig' 'purelin'},'trainlm');
T=[-0.03 3.5 -0.9 0.65 3.0];
net=train(net,P,T);
A=sim(net,P);
```

运行结果如图 3-45 所示。

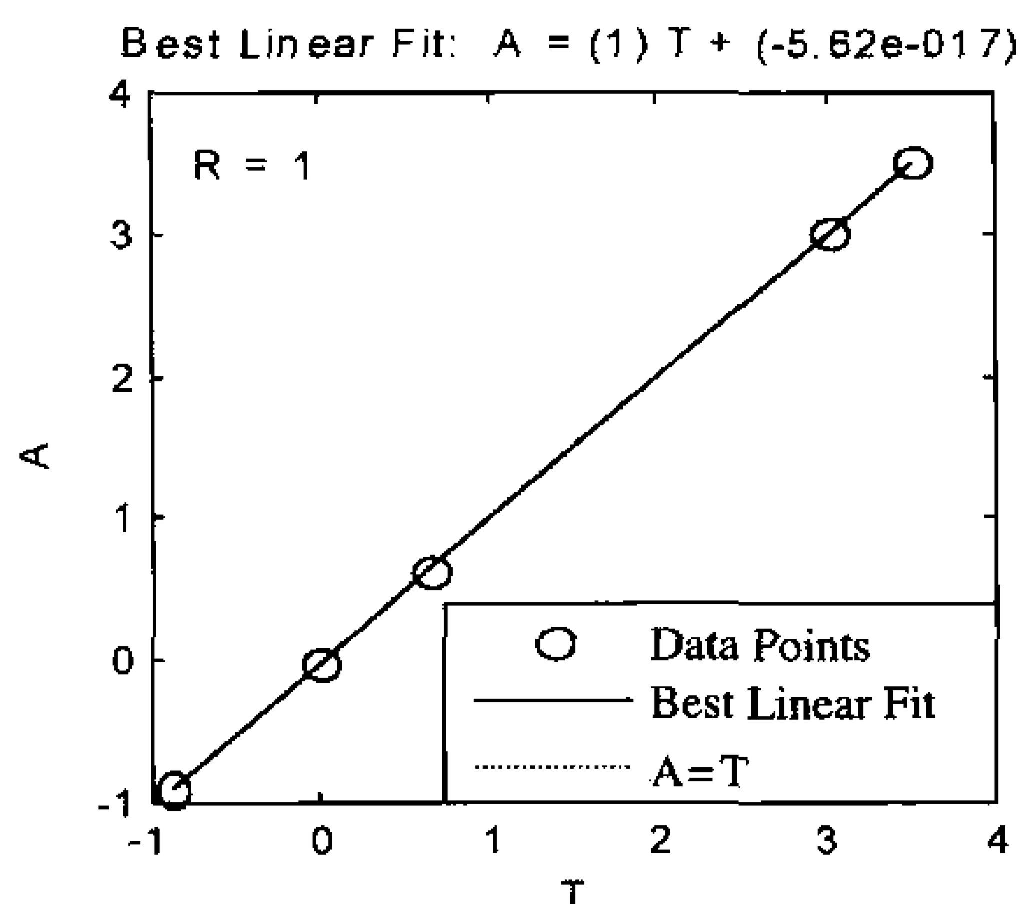


图 3-45　线性回归分析结果

6．prepca

应用：对输入数据进行主元分析。

格式：[Ptrans, TransMat]=prepca (P, min_frac)

解析：prepca 函数用来对输入数据矩阵 P 进行主元分析变换，该变换可以消除各输入矢量间的相关性。在变换后的矩阵中，部分矢量占用原始数据矩阵的大部分能量，保存了原始数据的大部分信息，因此变换后应予以保留；而有些矢量只占有原始数据的小部分能量，因此可以在变换后省略。

函数的输入 P 为原始数据矩阵；min_frac 为最小能量比例系数，当变换后矩阵中某一矢量的能量在原始数据总能量中所占的比例小于该系数时，这一矢量将被省略。函数返回经过主元分析的矩阵 Ptrans 和变换时使用的矩阵 TransMat。在该函数算法中都假定原始数据具有零均值，因此要首先利用函数 prestd 对数据进行归一化。

【例 3-109】对以下数据进行主元分析。

```
p=[-1.5 -0.58 0.21 -0.96;-2.2 -0.87 0.34 -1.4;-2.5 -0.38 0.44 -1.06];
pn=prestd(p);
[Ptrans,TransMat]=prepca(pn,0.01)
Ptrans =
     2.0218    -0.4191    -2.0912     0.4885
    -0.1787     0.1927    -0.1740     0.1601
TransMat =
    -0.5797    -0.5791    -0.5732
    -0.3770    -0.4330     0.8187
```

7．trastd

应用：利用预先计算的均值和方差对数据进行变换。

格式：[Pn]=trastd (P, meanp, stdp)

解析：本函数利用 prestd 函数对样本数据进行归一化时得到的均值和方差对新的输入数据

进行变换。表达式为

$$P=Pn*stdp+meanp$$

上式中的 P 和 Pn 分别为变换前、后的输入数据，meanp 和 stdp 分别为 prestd 函数找到的均值和方差。如果网络训练时所用的是归一化样本数据，那么以后使用网络时所用的新输入数据也应该和样本数据一样接受相同的预处理，这时就要用到 trastd 函数。

【例 3-110】利用级联前向网络对下列归一化样本数据进行学习，并用新的输入数据检验网络。

```
P=[-1.92 2.37 -1.48 0.86 3.12];
T=[-0.39 3.6 -0.95 0.75 3.2];
[Pn,minp,maxp,Tn,mint,maxt]=prestd(P,T);
net=newff(minmax(P),[5 1],{'tansig' 'purelin'},'trainlm');
net=train(net,Pn,Tn);
P2=[2 -2];
P2n=trastd(P2,minp,maxp);
A2n=sim(net,P2n);
A2=poststd(A2n,mint,maxt)
```

运行结果为

```
A2 =
      3.5470     -0.2676
```

训练过程如图 3-46 所示。

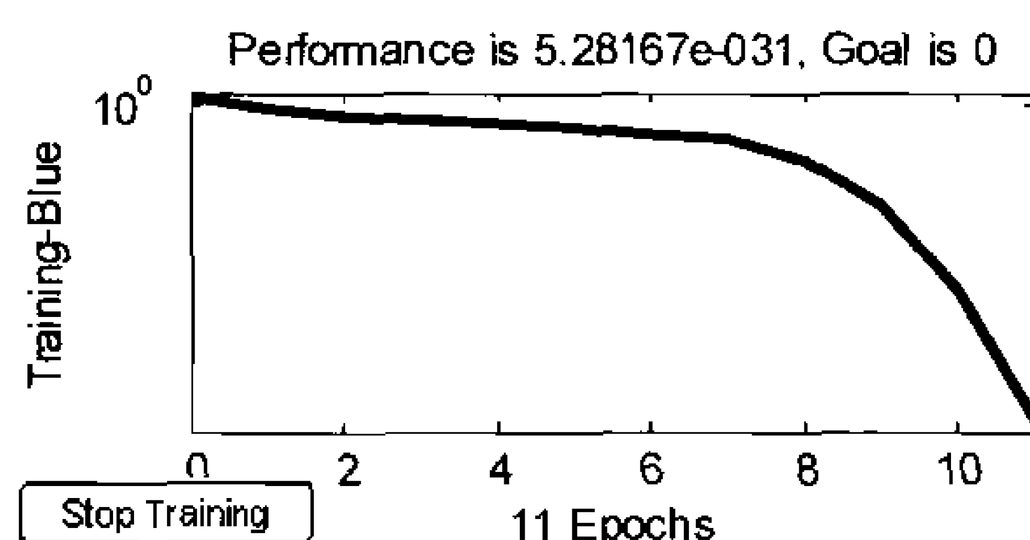


图 3-46　训练过程

8. trapca

应用：利用预先计算的主元分析矩阵对数据进行变换。

格式：[Ptrans]=trapca (P, TransMat)

解析：本函数利用 prepca 函数对样本数据进行主元分析时得到的变换矩阵 TransMat 对新的输入数据 P 进行变换。如果网络训练时所用的是经过主元分析的样本数据，那么以后使用网络时所用的新输入数据也应该和样本数据一样接受相同的预处理，这时就要用到 trapca 函数。

【例 3-111】利用级联前向网络对下列经过主元分析的样本数据进行学习，并用新的输入数据检验网络。

```
P=[-1.5 -0.58 0.21 -0.96 2.29;-2.2 -0.87 0.34 -1.4 -1.2];
T=[-0.39 3.6 -0.95 0.75 3.2];
[Pn,meanp,stdp,Tn,meant,stdt]=prestd(P,T);
[ptrans,transMat]=prepca(Pn,0.02);
net=newff(minmax(ptrans),[5 1],{'tansig' 'purelin'},'trainlm');
net=train(net,ptrans,Tn);
P2=[2.5 -1.8;-0.9 -2];
P2n=trastd(P2,meanp,stdp);
```

```
P2trans=trapca(P2n,transMat);
A2n=sim(net,P2trans);
A2=poststd(A2n,meant,stdt)
```

运行结果为

```
A2 =
     2.6182    -0.6545
```

9．tramnmx

应用： 利用预先计算的最大值和最小值对数据进行变换。

格式： [Pn]=tramnmx (P, minp, maxp)

解析： 本函数利用 premnmx 函数对样本数据进行归一化时得到的最小值和最大值对新的输入数据进行变换。表达式为

$$Pn=2*(P-minp)/(maxp-minp)-1$$

上式中的 P 和 Pn 分别为变换前、后的输入数据，maxp 和 minp 分别为 premnmx 函数找到的最大值和最小值。如果网络训练时所用的是归一化样本数据，那么以后使用网络时所用的新输入数据也应该和样本数据一样接受相同的预处理，这时就要用到 tramnmx 函数。

【例 3-112】利用级联前向网络对下列归一化样本数据进行学习，并用新的输入数据检验网络。

```
P=[-1.5 -0.58 0.21 -0.96 2.29];
T=[-0.39 3.6 -0.95 0.75 3.2];
[Pn,minp,maxp,Tn,mint,maxt]=premnmx(P,T);
net=newff(minmax(Pn),[5 1],{'tansig' 'purelin'},'trainlm');
net=train(net,Pn,Tn);
P2=[2 -2];
P2n=tramnmx(P2,minp,maxp);
A2n=sim(net,P2n);
A2=postmnmx(A2n,mint,maxt)
```

运行窗口输出结果为

```
A2 =
     1.2015    -0.4051
```

第 4 章　前向型神经网络及 MATLAB 应用举例

这一章首先将详细介绍前向型神经网络。前向型神经网络是这样的一类网络，它在计算输出值的过程中，输入值从输入层单元向前逐层传播，经过隐藏层最后到达输出层得到输出。前向网络第一层的单元与第二层所有的单元相连，第二层又与其上一层单元相连，同一层中的各单元之间没有连接。前向网络中神经元的激发函数，可采用线性硬阈值函数或单元上升的非线性函数等来表示。在训练过程中，它们的调节权值的算法都是采用有教师的 delta 学习规则。

对于前馈网络，根据神经元的传递函数不同，以及学习算法和网络结构上的区别，可以细分类感知器网络、线性网络、BP 网络、径向基网络及 GMDH 网络等不同的网络模型。本章将针对以上网络类型，对其理论基础、训练算法进行简单说明，并重点介绍如何利用 MATLAB R2007 神经网络工具箱实现这些网络。

4.1　感知器

感知器模型是由美国学者 F.Rosenblatt 于 1958 年提出的。它与 MP 模型的不同之处是它假定神经元的突触权值是可变的，这样就可以进行学习。感知器模型在神经网络研究中着重要的意义和地位，因为感知器模型包含了自组织、自学习的思想。

4.1.1　单层感知器模型

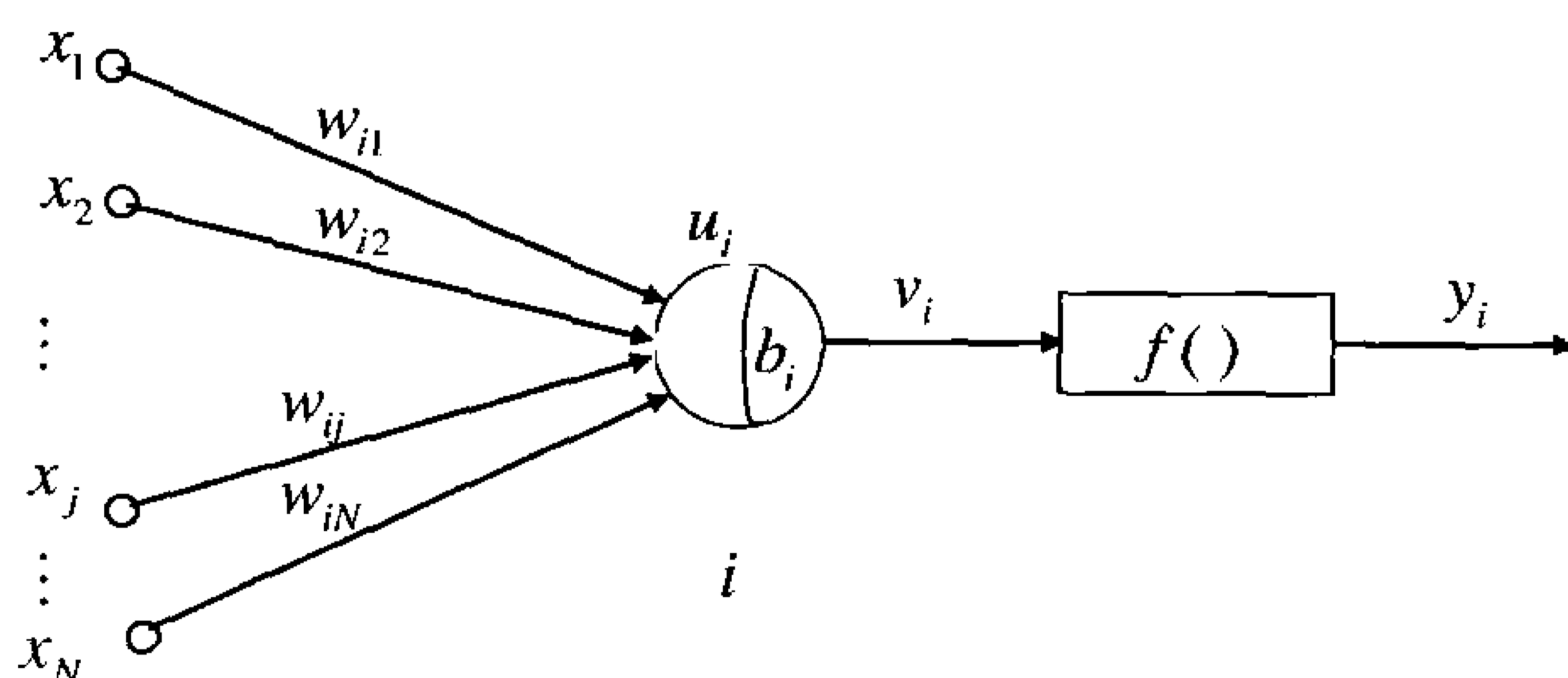


图 4-1　单层感知器模型

单层感知器模型如图 4-1 所示，它包括一个线性的累加器和一个二值阈值元件，同时还有一个外部偏差 b。线性累加器的输出作为二值阈值元件的输入，这样当二值阈值元件的输入是正数时，神经元就产生输出+1，反之，若其输入是负数，则产生输出-1。即

$$y = \mathrm{Sgn}\left(\sum_{j=1}^{m} w_{ij}x_j + b\right) \tag{4-1}$$

$$y = \begin{cases} +1 & 若\left(\sum_{j=1}^{m} w_{ij}x_j + b\right) \geqslant 0 \\ -1 & 若\left(\sum_{j=1}^{m} w_{ij}x_j + b\right) < 0 \end{cases} \tag{4-2}$$

使用单层感知器的目的就是让其对外部输入 $x_1, x_2, \cdots, x_m$ 进行识别分类，单层感知器可将外部输入分为两类：l_1 和 l_2。当感知器的输出为+1 时，我们认为输入 $x_1, x_2, \cdots, x_m$ 属于 l_1 类，当感知器的输出为-1 时，认为输入 $x_1, x_2, \cdots, x_m$ 属于 l_2 类，从而实现两类目标的识别。在 m 维信号空间，单层感知器进行模式识别的判决超平面由下式决定

$$\sum_{j=1}^{m} w_{ij} x_j + b = 0 \tag{4-3}$$

图 4-2 给出了一种只有两个输入 x_1 和 x_2 的判决超平面的情况，它的判决边界是直线：$w_1 x_1 + w_2 x_2 + b = 0$。

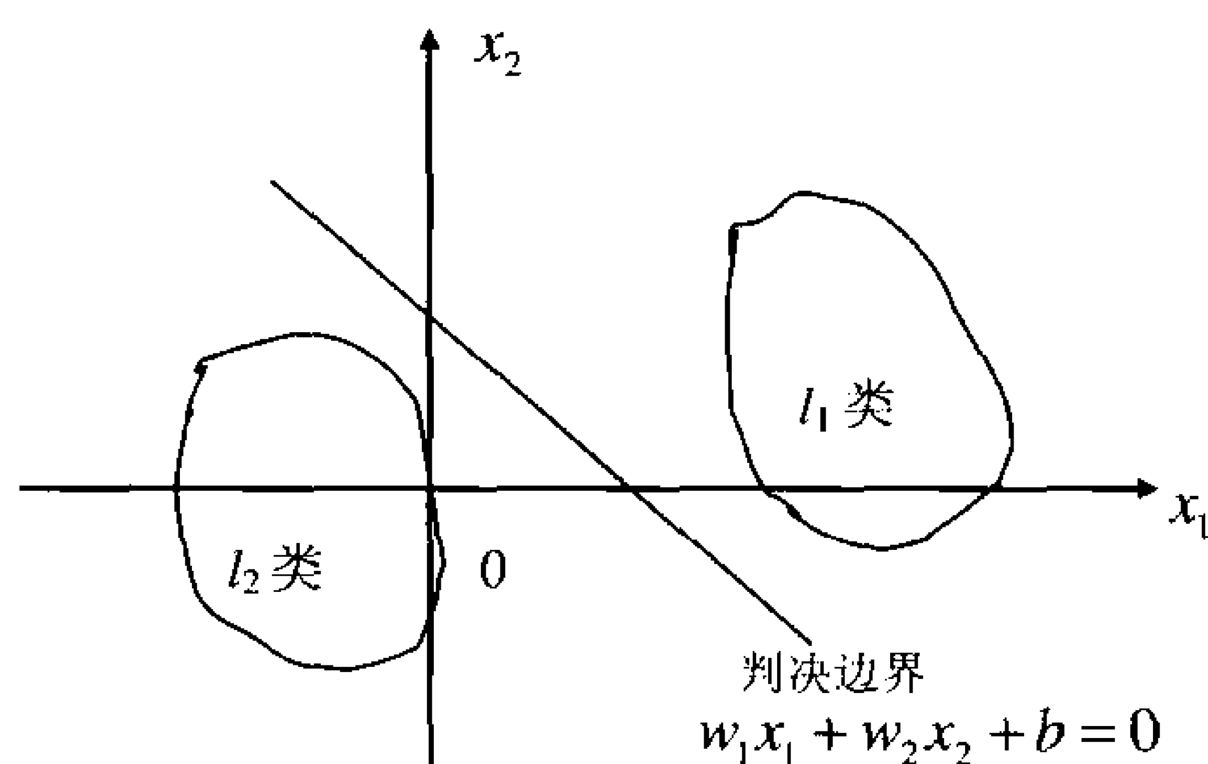


图 4-2 两类模式识别的判定问题

决定判决边界是直线的主要参数是 w_1 和 w_2，通过合适的学习算法可训练出满意的 w_1 和 w_2。

4.1.2 单层感知器的学习算法

单层感知器对权值向量的学习算法基于迭代的思想，通常是采用纠错学习规则的学习算法。

为方便起见，将偏差 $\boldsymbol{b}$ 作为神经元突触权值向量的第一个分量加到权值向量中去，那么对应的输入向量也应增加一项，可设输入向量的第一个分量固定为+1，这样输入向量和权值向量可分别写成如下的形式

$$\boldsymbol{X}(n) = [+1, x_1(n), x_2(n), \cdots, x_m(n)]^{\mathrm{T}} \tag{4-4}$$

$$\boldsymbol{W}(n) = [b(n), w_1(n), w_2(n), \cdots, w_m(n)]^{\mathrm{T}} \tag{4-5}$$

其中，变量 n 表示迭代次数，$b(n)$ 可用 $w_0(n)$ 表示，则二值阈值元件的输入可重新写为

$$v = \sum_{j=0}^{m} w_j(n) x_j(n) = \boldsymbol{W}^{\mathrm{T}}(n)\boldsymbol{X}(n) \tag{4-6}$$

令式（4-6）等于零，即 $\boldsymbol{W}^{\mathrm{T}}\boldsymbol{X} = 0$，可得在 m 维信号空间的单层感知器的判决超平面。

学习算法如下。

（1）设置变量和参量。

$\boldsymbol{X}(n) = [+1, x_1(n), x_2(n), \cdots, x_m(n)]^{\mathrm{T}}$，为输入向量，或称训练样本。

$\boldsymbol{W}(n) = [b(n), w_1(n), w_2(n), \cdots, w_m(n)]^{\mathrm{T}}$，为权值向量。

式中，$b(n)$为偏差，$y(n)$为实际输出，$d(n)$为期望输出，η 为学习速率，n 为迭代次数。

（2）初始化，赋给 $w_j(0)$ 一个较小的随机非零值，$n=0$。

（3）对于一组输入样本 $\boldsymbol{X}(n)=[+1,x_1(n),x_2(n),\cdots,x_m(n)]^{\mathrm{T}}$，指定它的期望输出 d（亦称之为导师信号）。如果 $\boldsymbol{X}\in l_1$ 则 $d=1$；如果 $\boldsymbol{X}\in l_2$ 则 $d=-1$。

（4）计算实际输出。

$$y(n)=\mathrm{Sgn}(\boldsymbol{W}^{\mathrm{T}}(n)\boldsymbol{X}(n)) \tag{4-7}$$

（5）调整感知器的权值向量。

$$\boldsymbol{W}(n+1)=\boldsymbol{W}(n)+\eta[d(n)-y(n)]\boldsymbol{X}(n) \tag{4-8}$$

（6）判断是否满足条件：若满足，算法结束；若不满足，将 n 值增加 1，转到第 3 步重新执行。

注意

以上学习算法的第 6 步需要判断是否满足条件，这里的条件可以是：误差小于设定的值 ε，即 $|d(n)-y(n)|<\varepsilon$；或者是权值的变化已很小，即 $|w(n+1)-w(n)|<\varepsilon$。另外，在实现过程中还应设定最大的迭代次数，以防止算法不收敛时，程序进入死循环。

在感知器学习算法中，重要的是引入了一个量化的期望输出 $d(n)$，其定义为

$$d(n)=\begin{cases}+1 & \text{如果}\boldsymbol{X}(n)\text{属于}l_1\\ -1 & \text{如果}\boldsymbol{X}(n)\text{属于}l_2\end{cases} \tag{4-9}$$

这样就可以采用纠错学习规则对权值向量进行逐步修正。

对于线性可分的两类模式，可以证明单层感知器的学习算法是收敛的，即通过学习调整突触权值可以得到合适的判决边界，正确区分两类模式，如图 4-3(a)所示。而对于线性不可分的两类模式，如图 4-3(b)所示，无法用一条直线区分两类模式。因而单层感知器的学习算法是不收敛的，即单层感知器无法正确区分线性不可分的两类模式。

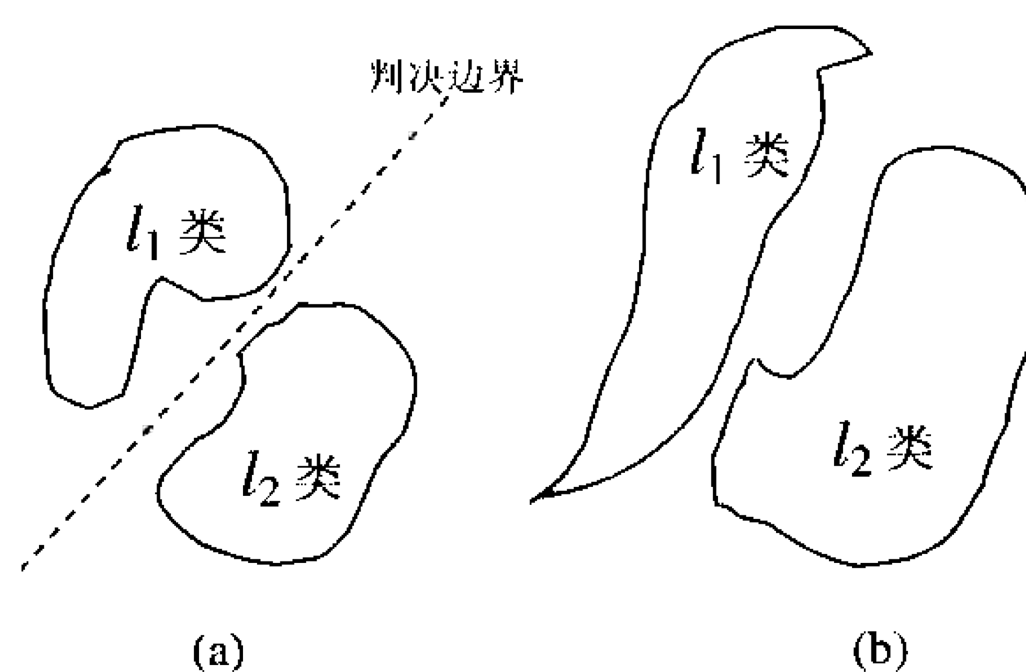


图 4-3　线性可分与不可分的问题

【例 4-1】试用单个感知器神经元完成下列分类，写出其训练的迭代过程，画出最终的分类示意图。

已知：$\left\{\boldsymbol{p}_1=\begin{bmatrix}2\\2\end{bmatrix},t_1=0\right\}$；$\left\{\boldsymbol{p}_2=\begin{bmatrix}1\\-2\end{bmatrix},t_2=1\right\}$；$\left\{\boldsymbol{p}_3=\begin{bmatrix}-2\\2\end{bmatrix},t_3=0\right\}$；$\left\{\boldsymbol{p}_4=\begin{bmatrix}-1\\0\end{bmatrix},t_4=1\right\}$

根据题意，神经元有两个输入量，传输函数为阈值型函数。于是以图 4-4 所示的感知器神经元完成分类。

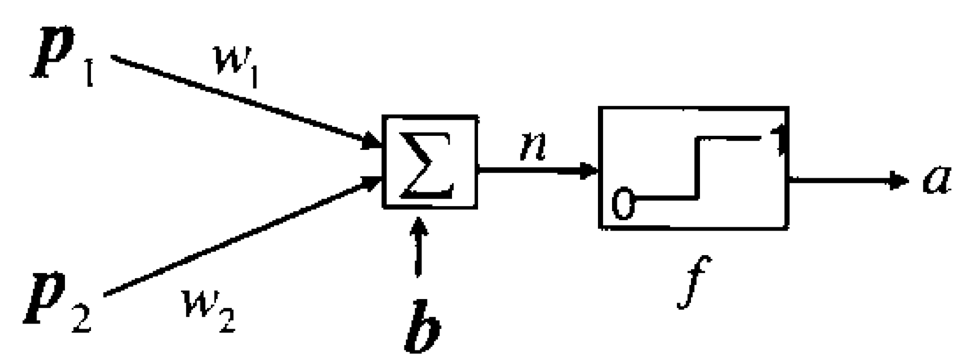


图 4-4　例 4-1 中的感知器神经元

（1）初始化：$\boldsymbol{W}(0)=[0\quad 0]$，$b(0)=0$

（2）第一次迭代。

$$a=f(n)=f[\boldsymbol{W}(0)\boldsymbol{p}_1+b(0)]=f\left([0\quad 0]\begin{bmatrix}2\\2\end{bmatrix}+0\right)=f(0)=1$$

$$e=t_1-a=0-1=-1$$

因为输出 a 不等于目标值 t_1，故需要调整权值和阈值。

（3）第二次迭代。以第二个输入样本作为输入向量，以调整后的权值和阈值进行计算。

$$a=f(n)=f[\boldsymbol{W}(1)\boldsymbol{p}_2+b(1)]=f\left([-2\quad -2]\begin{bmatrix}1\\-2\end{bmatrix}+(-1)\right)=f(1)=1$$

$$e=t_2-a=1-1=0$$

因为输出 a 等于目标值 t_2，所以无须调整权值和阈值。

$$\boldsymbol{W}(2)=\boldsymbol{W}(1)=[-2\quad -2]$$
$$b(2)=b(1)=-1$$

（4）第三次迭代，以第三个输入样本作为输入向量，以 $\boldsymbol{W}(2)$、$b(2)$ 进行计算。

$$a=f(n)=f[\boldsymbol{W}(2)\boldsymbol{p}_3+b(2)]=f\left([-2\quad -2]\begin{bmatrix}-2\\2\end{bmatrix}+(-1)\right)=f(-1)=0$$

$$e=t_3-a=0-0=0$$

因为输出 a 等于目标值 t_3，所以无须调整权值和阈值。

$$\boldsymbol{W}(3)=\boldsymbol{W}(2)=[-2\quad -2]$$
$$b(3)=b(2)=-1$$

（5）第四次迭代。以第四个输入样本作为输入向量，以 $\boldsymbol{W}(3)$、$b(3)$ 进行计算。

$$a=f(n)=f[\boldsymbol{W}(3)\boldsymbol{p}_4+b(3)]=f\left([-2\quad -2]\begin{bmatrix}-1\\0\end{bmatrix}+(-1)\right)=f(1)=1$$

$$e=t_4-a=1-1=0$$

因为输出 a 等于目标值 t_4，所以无须调整权值和阈值。

$$\boldsymbol{W}(4)=\boldsymbol{W}(3)=[-2\quad -2]$$
$$b(4)=b(3)=-1$$

（6）以后各次迭代又从第一个输入样本开始，作为输入向量，以前一次的权值和阈值进行计算，直到调整后的权值和阈值对所有的输入样本其输出的误差均为零为止。例如进行第五次迭代。

$$a=f(n)=f[\boldsymbol{W}(4)\boldsymbol{p}_5+b(4)]=f\left([-2\quad -2]\begin{bmatrix}2\\2\end{bmatrix}+(-1)\right)=f(9)=0$$

$$e=t_5-a=0-0=0$$

因为输出 a 等于目标值 t_4，所以无须调整权值和阈值。

$$\boldsymbol{W}(5)=\boldsymbol{W}(4)=[-2\quad -2]$$
$$b(5)=b(4)=-1$$

可以看出 $\boldsymbol{W}=[-2\quad -2]$，$b=-1$ 对所有的输入样本，其输出误差均为零，所以为最终调整后的权值和阈值。

（7）因为 $n>0$ 时，$a=1$；$n\leqslant 0$ 时，$a=0$，所以以 $n=0$ 作为边界。于是可以根据训练后的结果画出分类示意图，如图 4-5 所示。

其边界由下列直线方程（边界方程）决定。

$$n=\boldsymbol{W}\boldsymbol{p}+b=[-2\quad -2]\begin{bmatrix}\boldsymbol{p}_1\\\boldsymbol{p}_2\end{bmatrix}+(-1)=-2\boldsymbol{p}_1-2\boldsymbol{p}_2-1=0$$

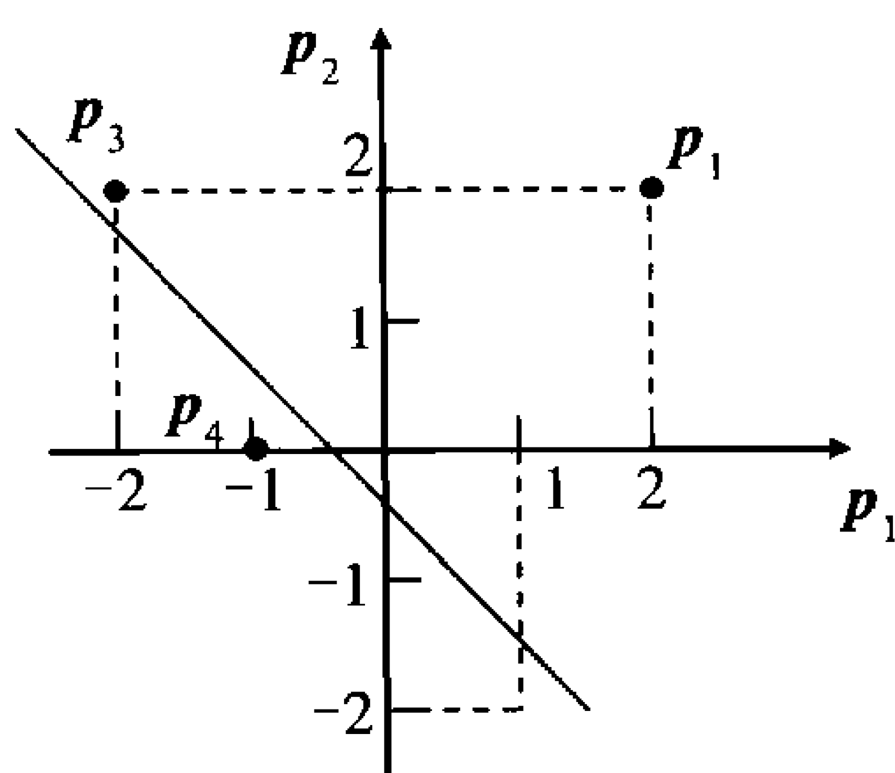


图 4-5　例 4-1 分类示意图

4.1.3　感知器的局限性

感知器神经网络的局限性在于：

（1）感知器神经网络的传输函数一般采用阈值函数，所以输出值只有两种（0 或 1，-1 或 1）；

（2）单层感知器网络只能用于解决线性可分的分类问题，而对线性不可分的分类问题无能为力；

（3）感知器学习算法只适于单层感知器网络，所以一般感知器网络都是单层的。

4.1.4 单层感知器神经网络的 MATLAB 仿真

1. 感知器神经网络设计的基本方法

单层感知器神经网络的 MATLAB 仿真程序设计主要包括以下几个方面。

1）以 newp 创建感知器神经网络

首先根据所要解决的问题，确定输入向量的取值范围和维数、网络层的神经元数目、传输函数和学习函数等；然后以单层感知器神经网络的创建函数 newp 创建网络。

2）以 train 训练创建网络

构造训练样本集，确定每个样本的输入向量和目标向量，调用函数 train 对网络进行训练，并根据训练的情况决定是否调整训练参数，以得到满足误差性能指标的神经网络，然后进行存储。

3）以 sim 对训练后的网络进行仿真

构造测试样本集，加载训练后的网络，调用函数 sim，以测试样本集进行仿真，查验网络的性能。

从以上过程可以看出，重要的感知器神经网络函数有 newp、train 和 sim，除此之外还涉及 init、trainc、dotprod、netsum、mae、plotpc、plotpv 等，这些函数的详解在第 3 章已经介绍过。

2. 单层感知器神经网络的应用举例

下面以应用实例说明单层感知器神经网络的 MATLAB 仿真程序设计。程序中涉及的其他 MATLAB 函数与命令，读者可自行参考第 3 章的内容。

【例 4-2】给定样本输入向量 ***P***、目标向量 ***T*** 及需要进行分类的输入向量组 ***Q***，设计一个单层感知器，对其进行分类。

MATLAB 代码如下。

```
P=[-0.6 -0.7 0.8;0.9 0 1];
T=[1 1 0];
net=newp([-1 1;-1 1],1);
%返回画线的句柄，下一次绘制分类线时将旧的删除
he=plotpc(net.iw{1},net.b{1});
%设置训练次数最大为 15
net.trainParam.epochs=15;
net=train(net,P,T);
%给定的输入向量
Q=[0.5 0.8 -0.2;-0.2 -0.6 0.6];
Y=sim(net,Q);
figure;
%绘制分类线
plotpv(Q,Y);
he=plotpc(net.iw{1},net.b{1},he)
```

运行结果如图 4-6 所示。由图 4-6 可见，设计的感知器对输入模式进行了成功的分类。

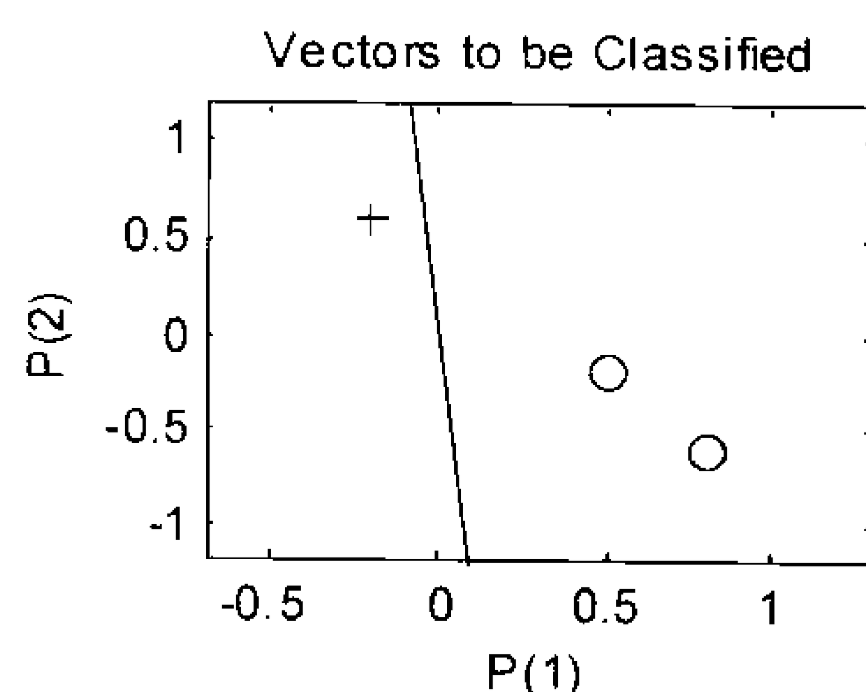


图 4-6 输入向量及分类线

感知器训练结果为

```
RAINC, Epoch 0/15
TRAINC, Epoch 3/15
TRAINC, Performance goal met.
he =
   154.0012
```

可见，经过 3 次训练后，网络目标误差达到要求，如果 4-7 所示。

下面给出一个线性不可分的例子。给定一个双元素的输入向量组 ***P***，每列都由 5 个元素组成，并给定一个目标向量 ***T***。利用以下代码将其绘制在一个平面上。运行结果如图 4-8 所示。

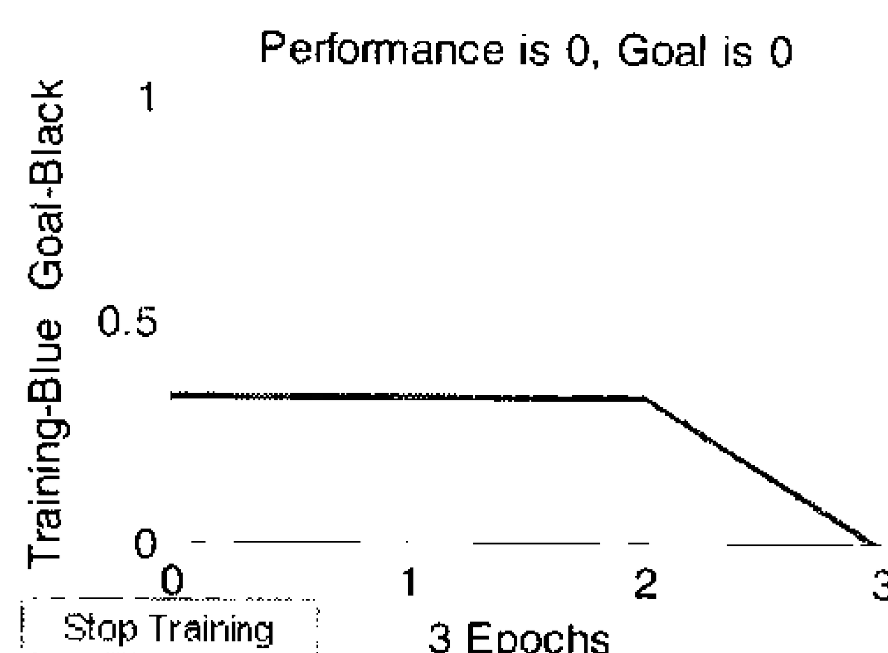


图 4-7 感知器训练过程

```
P=[-0.5 -0.5 0.3 -0.1 -0.8;-0.5 0.5 -0.5 1 0];
T=[1 1 0 0 0];
plotpv(P,T)
```

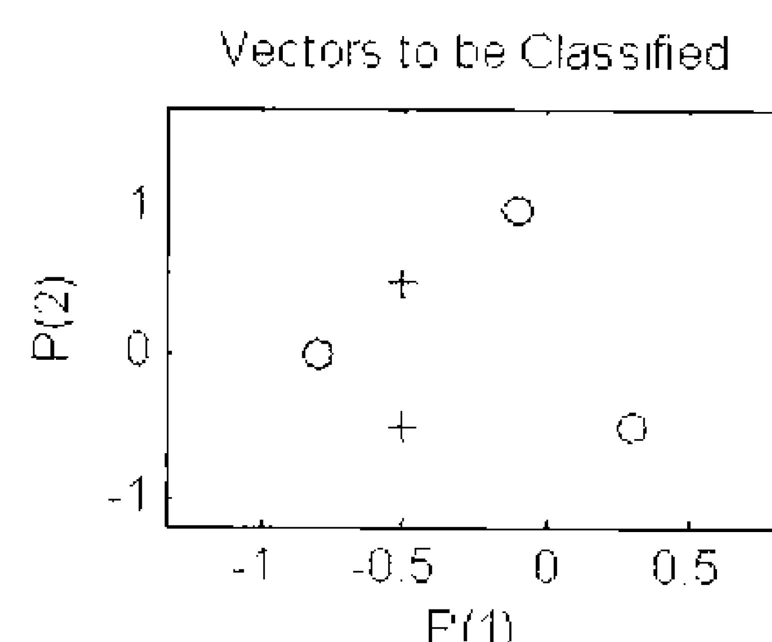


图 4-8 样本点的分布及相应的类别

接下来尝试设计一个感知器，该感知器必须对输入向量 ***P*** 进行准确的分类。利用函数 newp 创建一个感知器。

```
net=newp([-35 1;-2 45],1);
```

在利用感知器进行分类之前，首先需要对感知器进行初始化，将其权值设定为 0。这样一来，任何输入都将产生同样的输出，感知器也就不能进行分类了。

感知器必须经过训练才能应用，在这里使用自适应函数 adapt 对其进行训练。自适应函数

的循环次数设定为 4 次，经过 24 次迭代后，训练停止。其 MATLAB 代码如下。

```
net.adaptParam.passes=4;
linehandle=plotpc(net.iw{1},net.b{1});
for a=1:24
    [net,Y,E]=adapt(net,P,T);
    linehandle=plotpc(net.iw{1},net.b{1},linehandle);
    drawnow;
end;
```

对输入样本分类的运行结果如图 4-9 所示。由图 4-9 可见，对于这种线性不可分的模式，利用单层感知器是无法正确分类的。

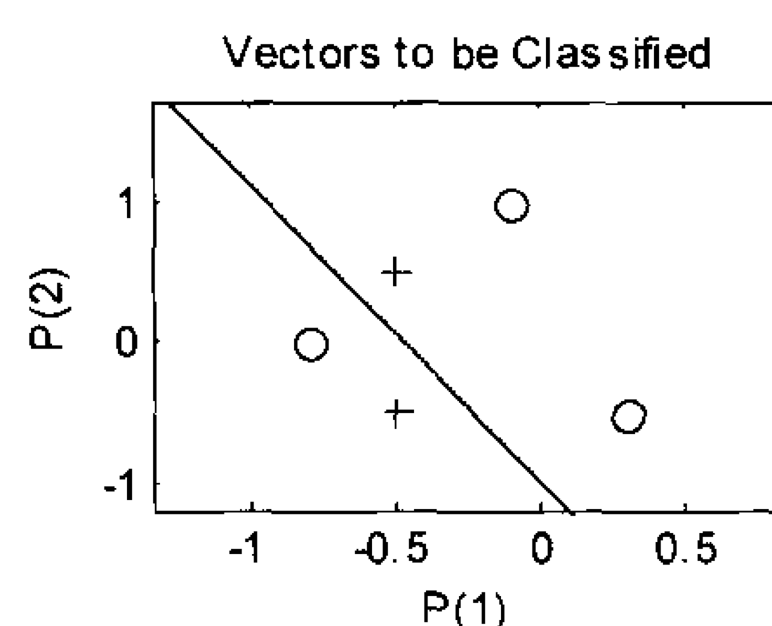


图 4-9　线性不可分样本的分类结果

【例 4-3】利用感知器实现三输入与门功能。三输入与门真值表如表 4-1 所示。

表 4-1　三输入与门真值表

样本数目	*A*	*B*	*C*	*Y*
1	0	0	0	0
2	0	0	1	0
3	0	1	0	0
4	0	1	1	0
5	1	0	0	0
6	1	0	1	0
7	1	1	0	0
8	1	1	1	1

利用感知器可以训练一个三输入与门功能。在这里使用两种方法训练神经网络：一种方法是采用全样本数据训练，而检验样本为输入样本增加 0.3 的误差，另一种方法是采用部分样本数据训练，剩余真值表数据作为检验样本数据。

训练方法一：采用全样本数据，检验样本增加 0.3。

```
%输入样本数据
P=[0 0 0 0 1 1 1 1;
   0 0 1 1 0 0 1 1;
   0 1 0 1 0 1 0 1];
%输入训练目标样本数据
T=[0 0 0 0 0 0 0 1];
%构建单神经元感知器网络
net=newp([repmat([-1 2],3,1)],1);
```

```
%训练单神经元感知器网络
E=1;
while (sse(E))
    [net,Y,E,Pf,Af,tr]=adapt(net,P,T);
end
%训练样本数据增加 0.3 的扰动
p=P+0.3;
A=sim(net,p)
```

输出结果为

```
A =
     0     0     0     0     0     0     0     1
```

可以看出仿真结果完全正确，训练好的单神经元感知器网络对输入样本数据具有一定的抗扰动能力，如果扰动过大，那么感知器网络将带有较大的误差，如扰动增加到 0.5 时，输入以下程序段。

```
p=P+0.5;
A=sim(net,p)
A =
     0     0     0     1     0     1     1     1
```

从结果中可以看出，此时单神经元感知器训练误差比较大，有三组样本数据输出错误。所以，从该实例中可以发现，神经网络具有一定的抗干扰能力，但扰动过大时，神经网络适应能力将下降。

训练方法二：采用部分样本数据作为训练样本。

```
%输入样本数据
P=[0 0 0 1 1 1;
   0 1 1 0 0 1;
   0 0 1 0 1 1];
%输入训练目标样本数据
T=[0 0 0 0 0 1];
%构建单神经元感知器网络
net=newp([repmat([-1 2],3,1)],1);
%训练单神经元感知器网络
E=1;
while (sse(E))
    [net,Y,E,Pf,Af,tr]=adapt(net,P,T);
end
%训练样本数据增加 0.3 的扰动
p=[0 1;0 1;1 0];
A=sim(net,p)
```

训练后输出结果为

```
A =
     0     1
```

从结果中可以看到，训练后的单神经元感知器网络对第 2 组检验数据输出错误。从这个实例可以看出，当利用单神经元感知器网络实现逻辑与门功能时，由于训练样本的数据空间不是

特别大，所以，应该采用全训练样本集，以确保感知器网络的正确性。

4.1.5 多层感知器神经网络及其 MATLAB 仿真

1．多层感知器神经网络的设计方法

单层感知器由于其结构和学习规则上的局限性，其应用也受到一定的限制，即它只能对线性可分的向量集合进行分类。为了解决线性不可分的输入向量的分类问题，可以增加网络层。

由于感知器神经网络学习规则的限制，它只能对单层感知器神经网络进行训练，那么，如何进行多层感知器神经网络的设计呢？这里提供了一种二层感知器神经网络的设计方法。

（1）把神经网络的第一层设计为随机感知器层，且不对它进行训练，而是随机初始化它的权值和阈值，当它接收各输入元素的值时，其输出也是随机的。但其权值和阈值一旦固定下来，对输入向量模式的映射也随之确定下来。

（2）以第一层的输出作为第二感知器层的输入，并对应输入模式，确定第二感知器层的目标向量，然后对第二感知器层进行训练。

（3）由于第一感知器层的输出是随机的，所以在训练过程中，整个网络可能达到训练误差性能指标，也可能达不到训练误差性能指标。所以，当达不到训练误差性能指标时，需要重新对随机感知器层的权值和阈值进行初始化赋值，可以将其初始化函数设置为随机函数，然后用 init 函数重新初始化。程序第一次运行的结果往往达不到设计要求，需要反复运行，直至达到要求为止。

2．多层感知器神经网络的应用举例

【例 4-4】单层感知器网络不能模拟异或函数，这里用二层感知器神经网络来实现。

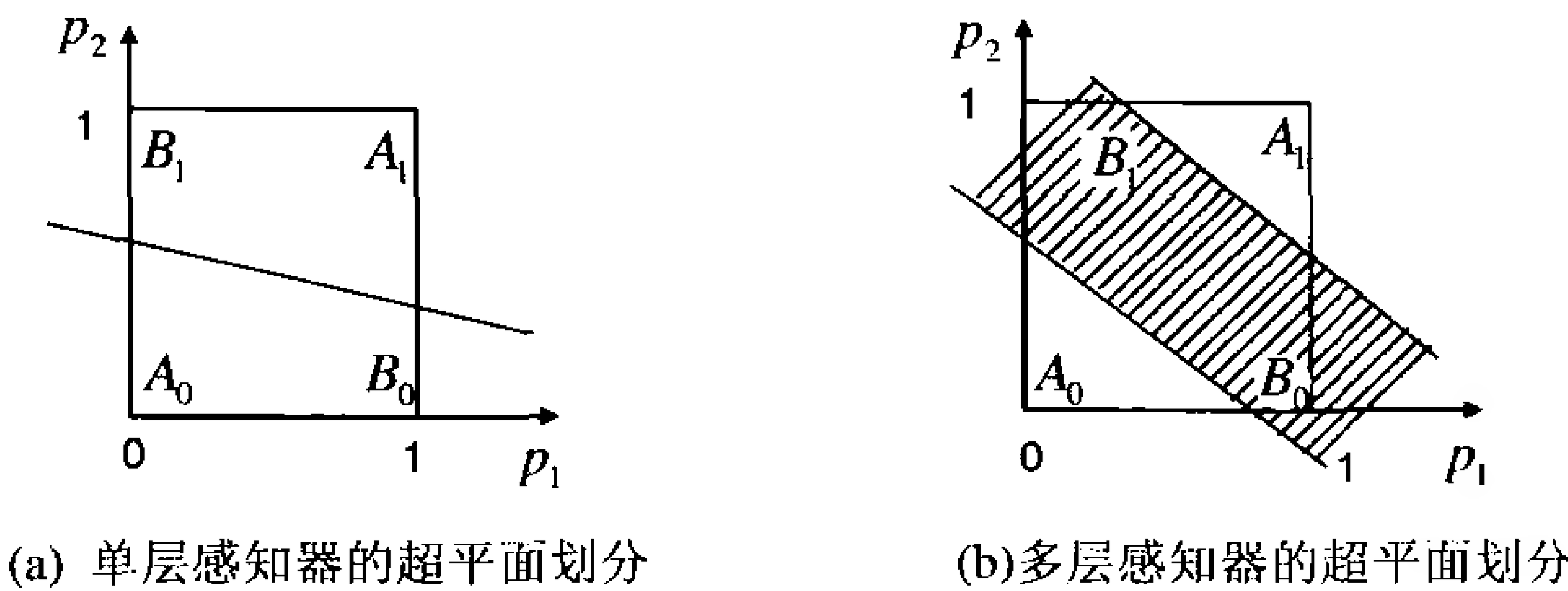
1）问题分析

异或真值表见表 4-2。

表 4-2 异或真值表

输入 p_1	输入 p_2	输出 a
0	0	0
1	0	1
0	1	1
1	1	0

若把异或问题看成 p_1-p_2 平面上的点，则点 $A_0(0,0)$、$A_1(1,1)$ 表示输出为 0 的两个点，$B_0(1,0)$、$B_1(0,1)$ 表示输出为 1 的两个点，如图 4-10 所示。



(a) 单层感知器的超平面划分　(b)多层感知器的超平面划分

图 4-10 异或问题的图形表示

从图 4-10 中可以看出，无论在平面上怎样用一条直线也不可能将输出为 0 和 1 的两种模式分开，而用两条直线就能将输出为 0 和 1 的两种模式分开。

2）神经网络的设计

根据以上分析，如果用两层感知器，每层感知器可以构成一条直线来划分，则可以解决模拟异或函数的问题。以图 4-11 所示的二层神经网络来实现，其中隐层为随机感知器层（net1），神经元数目设计为 3，其权值和阈值是随机的，它的输出作为输出层（分类层）的输入；输出层为感知器层（net2），其神经元数为 1，这里仅对该层进行训练。

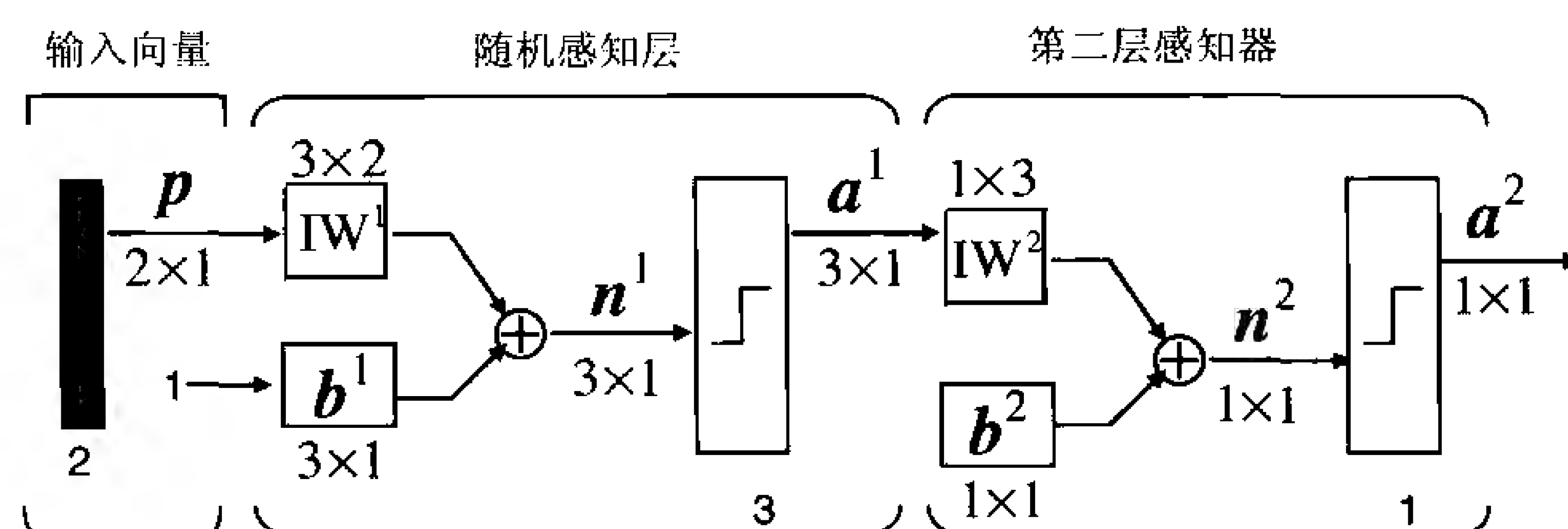


图 4-11　例 4-4 的二层感知器神经网络模型

3）多层感知器神经网络的 MATLAB 实现

代码如下。

```
clear all;
%初始化随机感知器层
PR1=[0 1;0 1];
net1=newp(PR1,3);
net1.inputweights{1}.initFcn='rands';
net1.biases{1}.initFcn='rands';
net1=init(net1);
IW1=net1.iw{1}
B1=net1.b{1}

%随机感知器层仿真量
P1=[0 0;0 1;1 0;1 1]';
[A1,Pf]=sim(net1,P1);

%初始化第二感知器层
PR2=[0 1;0 1;0 1];
net2=newp(PR2,1);

%训练第二感知器层
net2.trainParam.epochs=10;
net2.trainParam.show=1;
P2=ones(3,4);
P2=P2.*A1;
T2=[0 1 1 0];
[net2,TR2]=train(net2,P2,T2);
epoch2=TR2.epoch
perf2=TR2.perf
IW2=net2.iw{1}
```

```
B2=net2.b{1}
%存储训练后的网络
save net34 net1 net2
```

因为随机感知器层的输出是随机的，所以整个网络可能达到训练误差指标，也可能达不到训练误差指标。因此，当达不到训练误差指标时，需要重新对随机感知器层的权值和阈值进行初始化赋值，在本例程序中，是通过将其初始化函数设置为随机函数，然后用 init 函数重新初始化来实现的。该程序第一次运行的结果可能达不到设计要求，需要反复运行，直至达到要求为止。正因为如此，每次训练得到的结果也不尽相同。

这样做是因为设计受到感知器的学习算法的限制，对于多层网络，当然有更好的学习算法，比如后面将要介绍的 BP 网络学习算法。

其中达到训练误差指标的一种运行结果如下。

```
IW1 =
     0.7200    -0.0069
     0.7073     0.7995
     0.1871     0.6433
B1 =
    -0.6983
     0.3958
    -0.2433

TRAINC, Epoch 0/10
TRAINC, Epoch 1/10
TRAINC, Epoch 2/10
TRAINC, Epoch 3/10
TRAINC, Epoch 4/10
TRAINC, Epoch 5/10
TRAINC, Epoch 6/10
TRAINC, Epoch 7/10
TRAINC, Epoch 8/10
TRAINC, Epoch 9/10
TRAINC, Epoch 10/10
TRAINC, Maximum epoch reached.

epoch2 =
      0      1      2      3      4      5      6      7      8      9     10
perf2 =
      0.5000      0.7500      0.7500      0.5000      0.5000      0.7500      1.0000      1.0000
0.2500      0.5000      0.5000
IW2 =
     -3      2     -2
B2 =
      2
```

运行后，训练误差性能曲线如图 4-12 所示。

下面为多层感知器神经网络仿真的 MATLAB 程序。

```
clear all;
%加载训练后的网络
load net34 net2

%随机感知器层仿真
P1=[0 0;0 1;1 0;1 1]';
A1=sim(net1,P1);

%输出感知器层仿真，并输出仿真结果
P2=ones(3,4);
P2=P2.*A1;
A2=sim(net2,P2)
```

输出仿真结果为

```
A2 =
     1     1     1     0
```

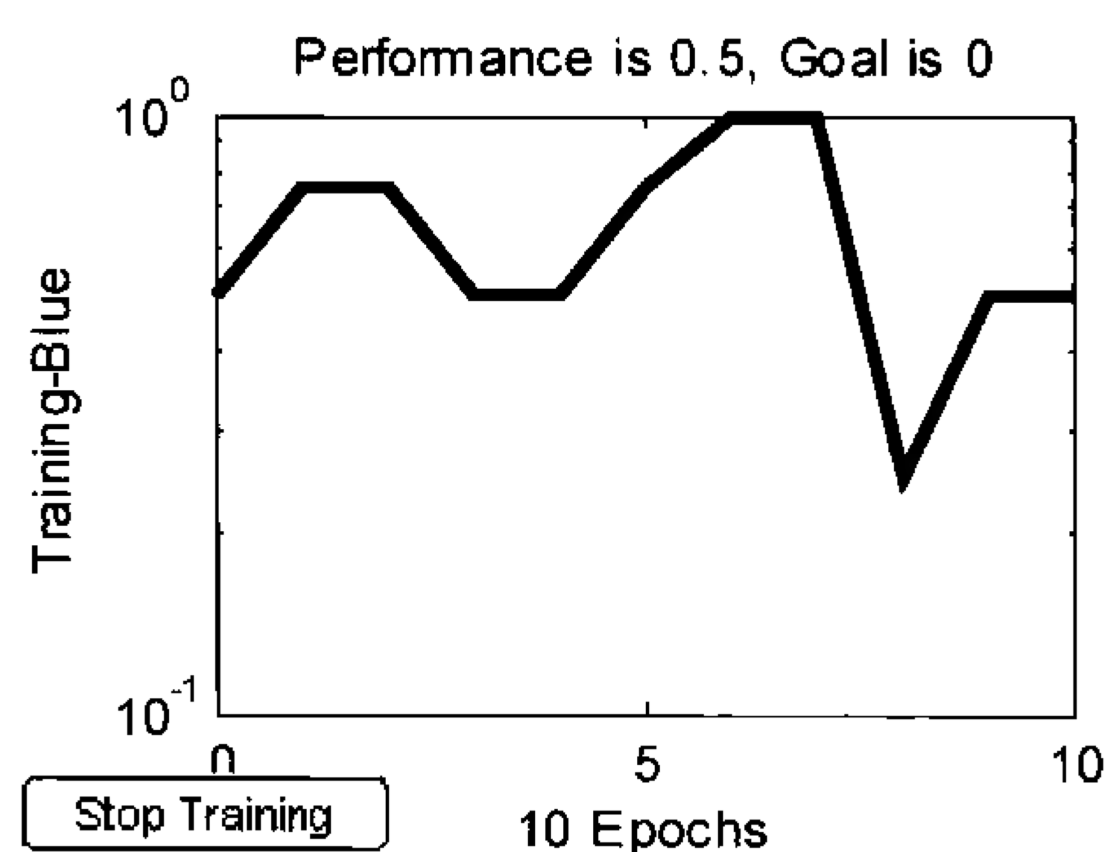


图 4-12　训练误差性能曲线

可以看出，所设计的网络可以正确模拟“异或”函数的功能。

4.1.6　用于线性分类问题的进一步讨论

我们可以把神经网络实现的功能看成是输入到输出的映射，如果把每一种不同的输入看成是一种输入模式，将其到输出的映射看成是输出响应模式，则输入到输出的映射就变成输入模式空间到输出模式空间的映射。这种输入模式到输出模式的映射，就是模式分类问题。

1．决策函数与决策边界

模式分类的基本内容是确定判决函数与决策边界。对于 C 类分类问题，按照判决规则可以把特征向量空间（或称模式空间）分成 C 个决策域。将划分决策域的边界称为决策边界，在数学上可以用解析形式将其表示成决策边界方程。用于表达决策规则的某些函数称为判决函数。判决函数与决策边界方程是密切相关的，而且它们都由相应的决策规则所确定。

神经网络用于模式分类，其决策函数为

$$f(n)=f(\boldsymbol{Wp}+\boldsymbol{b}) \tag{4-10}$$

决策边界由相应的决策函数和决策规则所确定。一般来说，当模式 $\boldsymbol{P}$ 为一维时，决策边界为一分界点；当 $\boldsymbol{P}$ 为二维时，决策边界为一直线；当 $\boldsymbol{P}$ 为三维时，决策边界为一平面；当 $\boldsymbol{P}$ 为 n（n>3）维时，决策边界为一超平面。图 4-13 画出了 n=1、2、3 的情况。

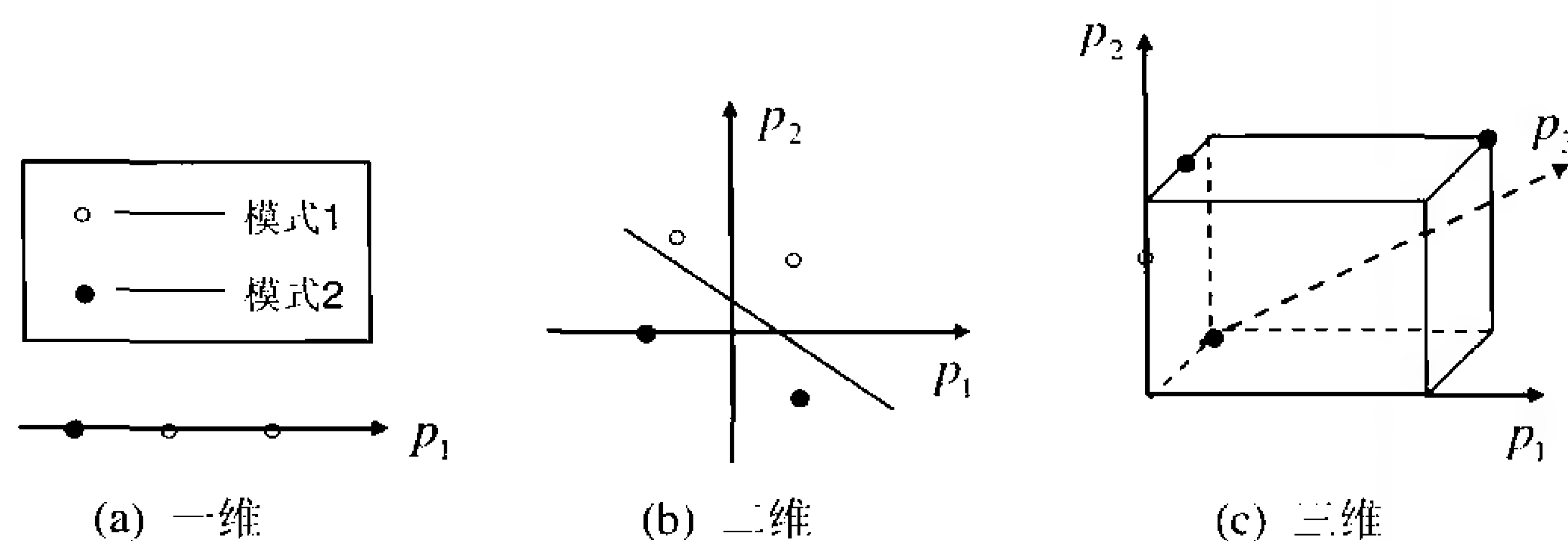


图 4-13　输入维数不同时的决策边界

在 MATLAB 神经网络工具箱中，可以用 plotpc 函数绘制 $R \leqslant 3$ 感知器神经网络的分类线，函数详细说明请参见第 3 章内容。

2. 感知器的决策函数与决策边界

感知器神经元的传输函数为阈值型函数，若传输函数为 hardlim 函数，则其决策函数为

$$f(u) = \text{hardlim}(\boldsymbol{Wp} + \boldsymbol{b}) = \begin{cases} 0, & \boldsymbol{Wp} + \boldsymbol{b} < 0 \\ 1, & \boldsymbol{Wp} + \boldsymbol{b} \geqslant 0 \end{cases} \tag{4-11}$$

其值只有 0 和 1 两种情况，所以决策边界由下列边界方程决定

$$\boldsymbol{Wp} + \boldsymbol{b} = 0 \tag{4-12}$$

单层感知器只有一个边界方程，且为线性方程，所以它只能进行线性分类。

【例 4-5】设计一个感知器神经网络，完成下列分类，以 MATLAB 编写仿真程序，并画出分类线。已知

$$\boldsymbol{p}_1 = \begin{bmatrix} 0.5 \\ -1 \end{bmatrix},\ t_1 = 0;\ \boldsymbol{p}_2 = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix},\ t_2 = 1;\ \boldsymbol{p}_3 = \begin{bmatrix} -1 \\ 0.5 \end{bmatrix},\ t_3 = 1;\ \boldsymbol{p}_4 = \begin{bmatrix} -1 \\ -1 \end{bmatrix},\ t_4 = 0$$

1）问题分析

输入向量有 2 个元素，取值范围为[-1，1]；输出向量有 1 个元素，是一个二值元素，取值为 0 或 1。由此可以确定单层感知器神经网络的结构：1 个输入向量，包括 2 个元素，1 个神经元，神经元的传输函数为 hardlim。

2）MATLAB 程序设计

```
clear all
%初始化感知器网络
PR=[-1 1;-1 1];
net=newp(PR,1);
%net.layers{1}.transferFcn='hardlims';
%训练感知器网络
P=[0.5 -1;1 0.5;-1 0.5;-1 -1]';
T=[0 1 1 0];
[net,TR]=train(net,P,T);
%神经网络仿真的 MATLAB 程序
%网络仿真
p=[0.5 -1;1 0.5;-1 0.5;-1 -1]'
A=sim(net,P)
```

```
%绘制网络的分类结果及分类线
V=[-2 2 -2 2];
plotpv(p,A,V);
plotpc(net.iw{1},net.b{1});
```

仿真结果如下。

```
p =
    0.5000    1.0000   -1.0000   -1.0000
   -1.0000    0.5000    0.5000   -1.0000
A =
     0     1     1     0
```

分类结果及分类线如图 4-14 所示。

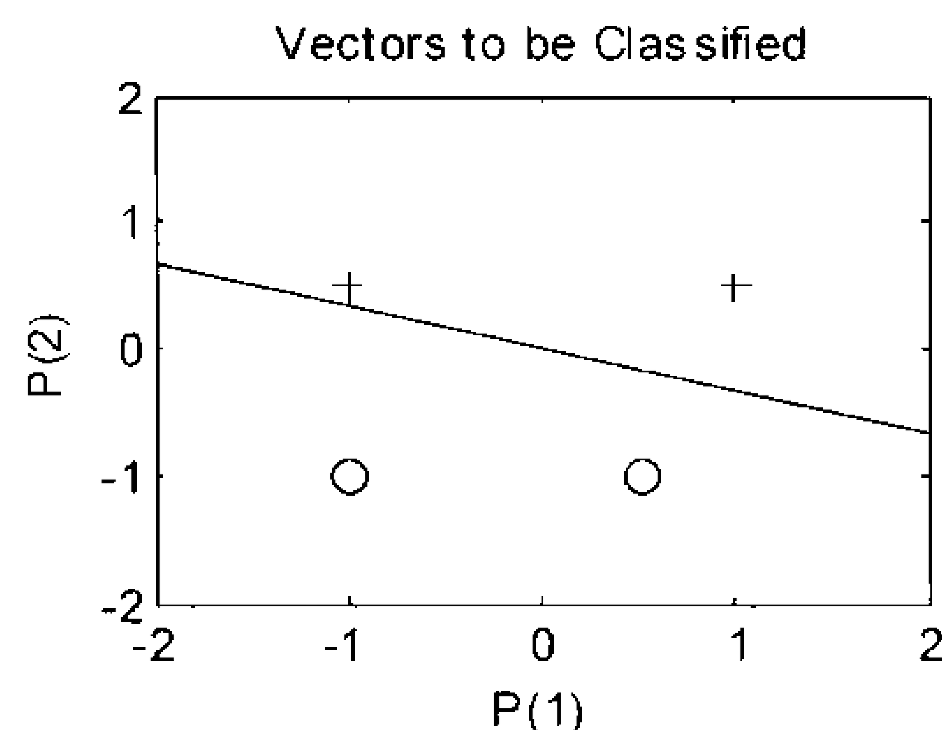


图 4-14　例 4-5 的分类结果及分类线

感知器是一种最简单的神经网络模型，它只能用于解决线性可分的问题，但它也是第一种具有训练算法的网络。本节通过对感知器的讨论，介绍了神经元、神经网络模型、神经网络训练与学习规则等基本知识，以及神经元和神经网络模型在 MATLAB NNET ToolBox 中的表示方法。用 MATLAB 对感知器神经网络进行仿真，最基本的 3 个函数是网络创建函数、网络训练函数和网络仿真函数。掌握感知器神经网络的基本知识及其仿真程序设计的一般方法是学习其他神经网络模型的基础。

4.2　线性神经网络

线性神经网络是最简单的一种神经网络，它由一个或多个线性神经元构成。1960 年由 B.Widrow 和 M.E.Hoff 提出的自适应线性单元（Adaline）网络是线性神经网络最早的典型代表。线性神经网络采用线性函数作为传递函数，因此其输出可以取任意值。线性神经网络可以采用基于最小二乘算法（LMS）的 Widrow-Hoff 学习规则来调节网络的权值和阈值，其收敛速度和精度都有较大的改进。此外，采用 newlind 函数还可以直接根据网络的输入矢量和目标矢量设计出期望的线性网络。和感知器神经网络一样，线性神经网络只能反映输入和输出样本矢量间的线性映射关系，它只能解决线性可分问题。线性神经网络在函数拟合、信号滤波、预测和控制等方面都有着广泛的应用。

4.2.1　线性神经网络结构

1．线性神经元模型

线性神经元模型的结构如图 4-15 所示。

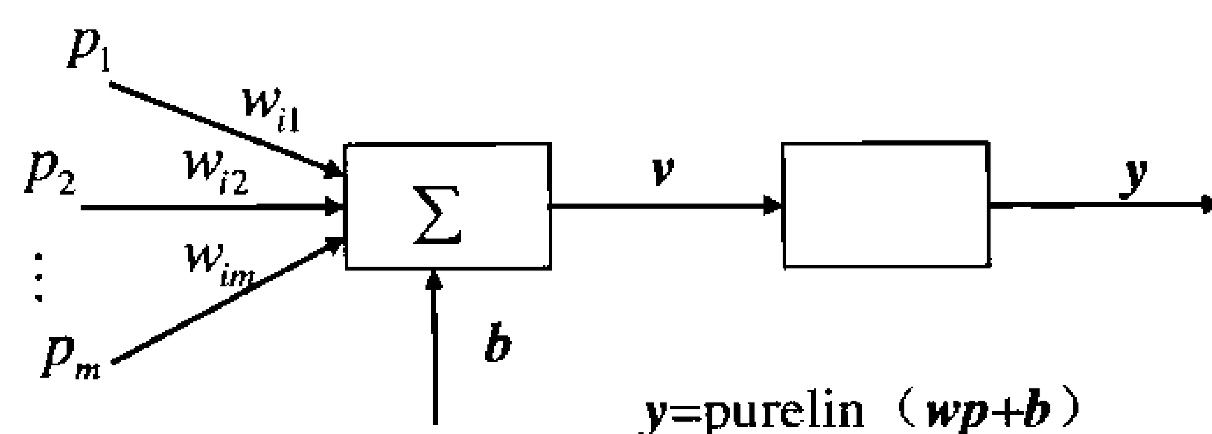


图 4-15 线性神经元模型

与感知器神经元不同的是，线性神经元采用的传递函数为线性函数 purelin，其他输入与输出之间是简单的纯比例关系。purelin 函数的具体应用请参见第 3 章。线性神经元的输出可以取任意值，其输入、输出关系为

$$\boldsymbol{a} = \text{purelin}(\boldsymbol{wp} + \boldsymbol{b}) = \boldsymbol{wp} + \boldsymbol{b} \tag{4-13}$$

2. 线性神经网络结构

图 4-16 给出了具有 R 维输入的单层（包含 S 个神经元）线性神经网络模型。

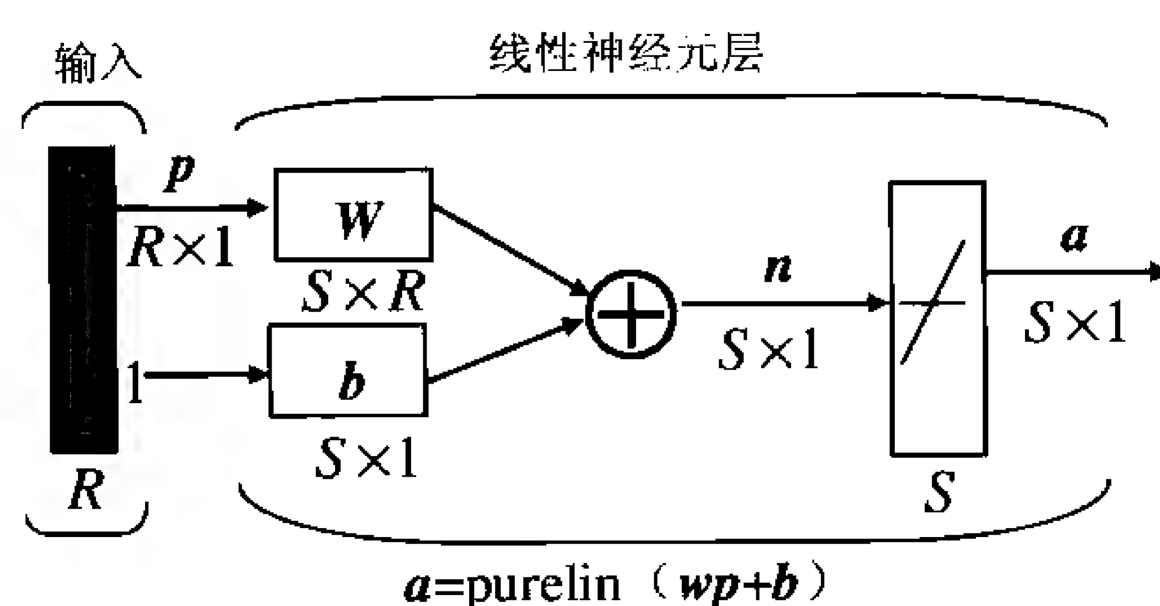


图 4-16 线性神经网络

4.2.2 线性神经网络设计

1. 线性神经网络的学习规则

线性神经网络权值和阈值的学习规则采用的是基于最小二乘原是的 Widrow-Hoff 学习算法。基于 Widrow-Hoff 学习算法的权值和阈值调节原理如下。

$$\boldsymbol{W}(k+1) = \boldsymbol{W}(k) + \Delta\boldsymbol{W}(k)\,,\quad \boldsymbol{b}(k+1) = \boldsymbol{b}(k) + \Delta\boldsymbol{b}(k) \tag{4-14}$$

其中

$$\Delta\boldsymbol{W}(k) = -\alpha\frac{\partial e^2(k)}{\partial \boldsymbol{W}} \qquad \Delta\boldsymbol{b}(k) = -\alpha\frac{\partial e^2(k)}{\partial \boldsymbol{b}} \tag{4-15}$$

由于

$$\frac{\partial e^2(k)}{\partial \boldsymbol{W}} = 2e(k)\frac{\partial e}{\partial \boldsymbol{W}} \qquad \frac{\partial e^2(k)}{\partial \boldsymbol{b}} = 2e(k)\frac{\partial e}{\partial \boldsymbol{b}} \tag{4-16}$$

且有

$$\frac{\partial e}{\partial \boldsymbol{W}} = \frac{\partial}{\partial \boldsymbol{W}}[\boldsymbol{t}(k) - (\boldsymbol{W}\times\boldsymbol{p}(k) + \boldsymbol{b})] = -\boldsymbol{p}(k) \qquad \frac{\partial e}{\partial \boldsymbol{b}} = -1 \tag{4-17}$$

所以，上述权值和阈值的调节公式可变为

$$\boldsymbol{W}(k+1) = \boldsymbol{W}(k) + 2\alpha e(k)\boldsymbol{p}^{\mathrm{T}}(k) \qquad \boldsymbol{b}(k+1) = \boldsymbol{b}(k) + 2\alpha e(k) \tag{4-18}$$

即

$$\boldsymbol{W}(k+1) = \boldsymbol{W}(k) + \eta e(k)\boldsymbol{p}^{\mathrm{T}}(k) \qquad \boldsymbol{b}(k+1) = \boldsymbol{b}(k) + \eta e(k) \tag{4-19}$$

其中，η 为学习速率。当 η 较大时，学习速率加快（η 取值过大时有可能使学习变得不稳定），反之亦然。

在 MATLAB 中，Widrow-Hoff 学习算法对应的学习函数为 learnwh。

2．线性神经网络的训练和仿真

对线性神经网络的训练可以调用 train 函数完成。利用 train 函数对线性神经网络进行训练实际上是根据所给出的“输入—目标”样本矢量集，调用神经网络生成时所定义的权值和阈值学习函数 learnwh 对网络不断进行调节，最终使网络输出接近目标输出的过程。下面的代码给出了一个典型的具有二维输入线性神经元的训练过程。

```
clear all;
P=[1 2 -1 0;2 2 0 -1];
T=[0 0 1 1];
net=newlin([-2 2;-2 2],1);
net.trainParam.goal=0.01;
[net,tr]=train(net,P,T);
```

训练中会显示如下信息。

```
TRAINB, Epoch 0/100, MSE 0.5/0.01.
TRAINB, Epoch 25/100, MSE 0.121667/0.01.
TRAINB, Epoch 50/100, MSE 0.0334739/0.01.
TRAINB, Epoch 75/100, MSE 0.012111/0.01.
TRAINB, Epoch 82/100, MSE 0.00983141/0.01.
TRAINB, Performance goal met.
```

可见，当训练到第 82 步时，网络性能达标。此时的权值和阈值矩阵为

```
>> W=net.iw{1,1}
W =
    -0.1854    -0.2112
>> b=net.b{1}
b =
     0.6973
```

利用 sim 等相关函数可以对训练好的线性神经网络进行仿真和误差分析。

```
>> a=sim(net,P)
a =
     0.0894    -0.0959     0.8827     0.9086
>> error=T-a
error =
    -0.0894     0.0959     0.1173     0.0914
>> m=mse(error)
m =
     0.0098
```

3．线性神经网络的直接设计法

与其他神经网络设计不同的是，线性神经网络可以根据输入和目标矢量直接设计出来。函数 newlind 无须经过训练，就可以直接设计出线性神经网络，使得网络实际输出与目标输出的平方和误差 SSE 为最小，其常用格式为

net=newlind (P,T)

4.2.3 自适应滤波线性神经网络

1. 自适应滤波线性神经网络结构

自适应滤波线性神经网络（简称自适应滤波网络）是线性神经网络的一种特殊形式，它与一般线性神经网络的不同之处在于自适应滤波网络引入了如图 4-17 所示的 TDL。

可见，自适应滤波网络的输入是由同一输入信号的若干延迟信号所构成的，即有

$$\begin{gathered}\boldsymbol{pd}_1(k)=\boldsymbol{p}(k)\\ \boldsymbol{pd}_2(k)=\boldsymbol{p}(k-1)\\ \vdots\\ \boldsymbol{pd}_N(k)=\boldsymbol{p}(k-N+1)\end{gathered}$$

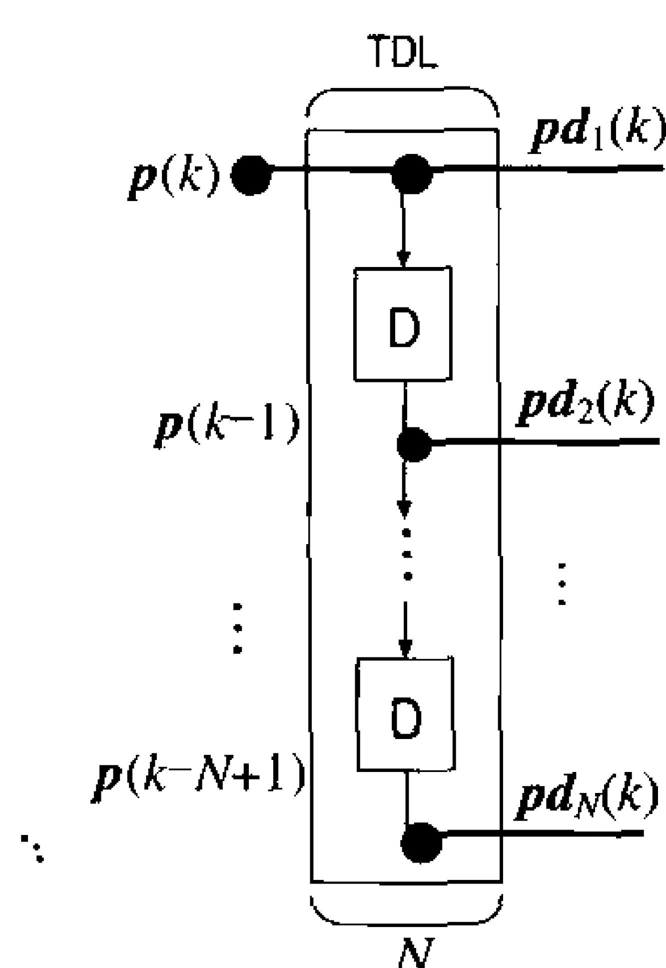


图 4-17　TDL 延迟链

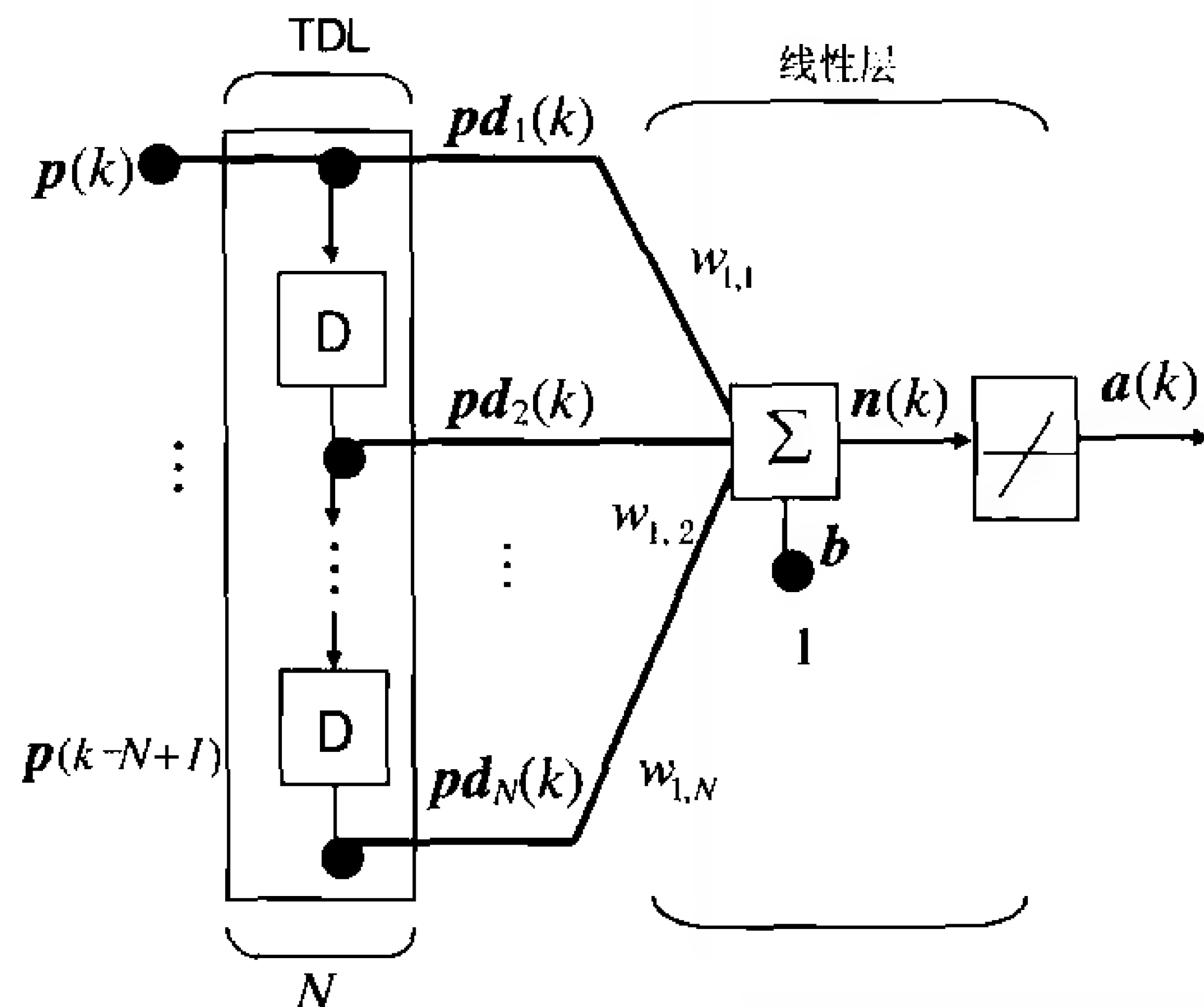


图 4-18　自适应滤波网络结构

一个典型的自适应滤波网络的结构如图 4-18 所示，其中，线性层只有一个神经元。该神经网络的输出为

$$\boldsymbol{a}(k)=\text{purelin}(\boldsymbol{wp}+\boldsymbol{b})=\sum_{i=1}^{N}w_{1,i}\boldsymbol{p}(k-i+1)+\boldsymbol{b} \tag{4-20}$$

可见，自适应滤波网络的输出是由输入信号 $\boldsymbol{p}(k)$ 及其延迟信号的线性组合构成的，这正好与数字信号处理领域所谓的有限冲击响应（FIR）滤波器的结构形式相吻合。

当然，自适应滤波网络的线性层也可以由多个线性神经元组成。一个典型的具有多个神经元的自适应滤波网络结构的缩略形式如图 4-19 所示。

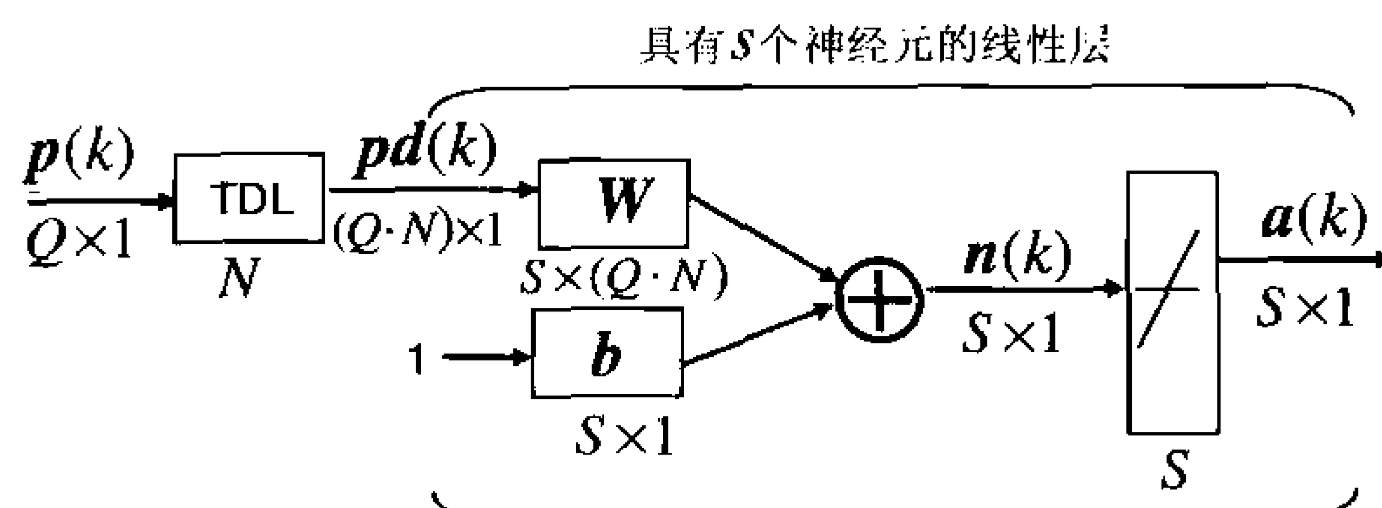


图 4-19　具有 S 个神经元的自适应滤波网络结构的缩略形式

迄今为止，自适应滤波网络已成为应用最为广泛的神经网络之一，它在信号滤波、预测与控制等领域都有着广泛的应用。

2．自适应滤波线性神经网络设计

自适应滤波网络的生成可以采用两种方式：一种是通过调用 newlin 函数直接生成带有延迟链的自适应滤波网络；另一种则是首先利用 newlin 函数生成不带延迟链的线性网络，然后通过网络重定义将延迟链加入预先生成的线性网络中。图 4-20 为单神经元自适应滤波网络的结构示意图。

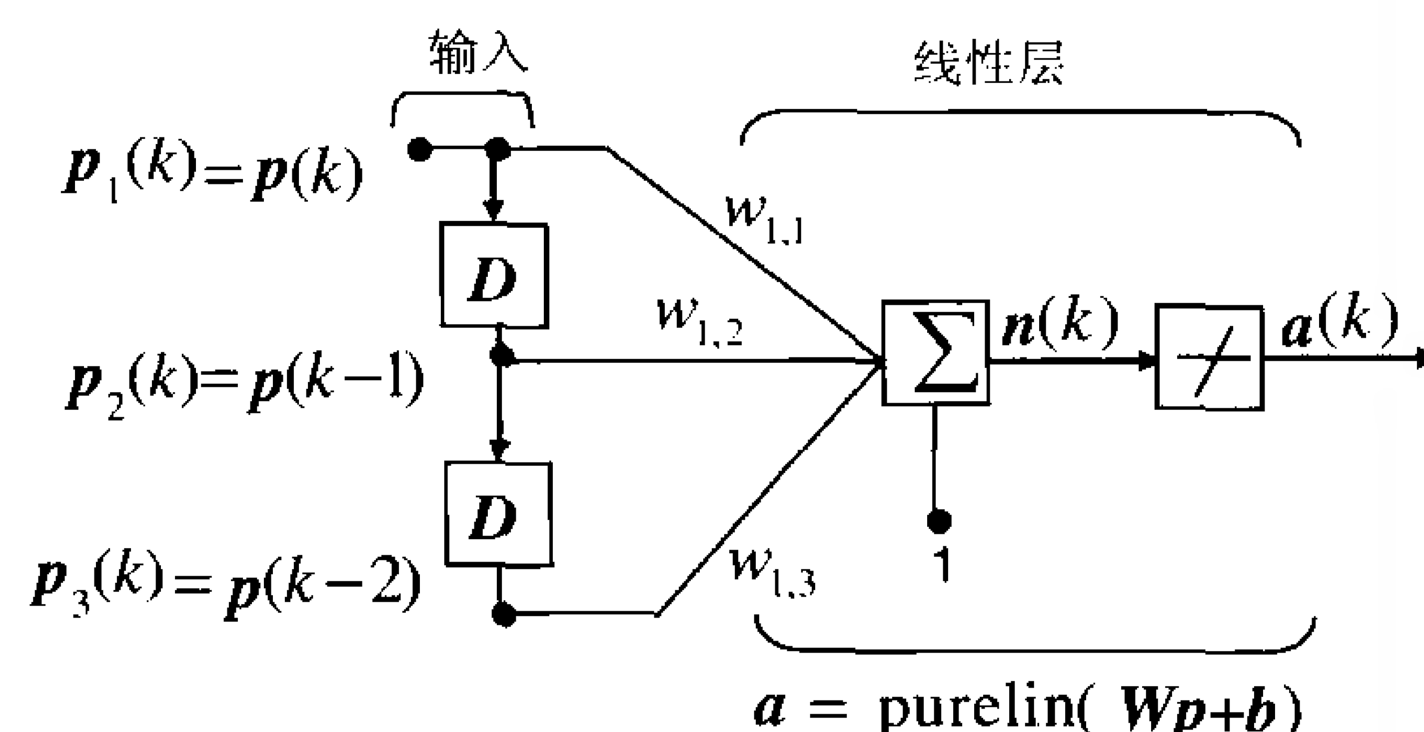


图 4-20　单神经元自适应滤波网络结构示意图

该自适应滤波网络可以采用如下两种方式来生成（假设网络输入范围为[0，10]）。

方式一：

```
net=newlin ([0，10]，1，[0 1 2]);
```

方式二：

```
net=newlin([0，10]，1);
net.inputWeights{1,1}.delays=[0 1 2];
```

其中，[0 1 2]中各元素分别表示自适应滤波网络各维输入所对应的延迟量。下面我们对所生成的自适应滤波网络分别进行初始化、仿真和训练。

自适应滤波网络的初始化与一般的线性神经网络基本相同，只是在初始化网络权值和阈值的同时，自适应滤波网络还要对延迟输入的初始值进行设置。

首先，对网络初始权值和阈值进行设置。

```
net=newlin([0,10],1,[0 1 2]);
net.iw{1,1}=[7 8 9];
net.b{1}=0;
```

其次，在对网络进行仿真之前需要对延迟输入 $p_2(k)$ 和 $p_3(k)$ 的初始值进行设置。

```
Pi={1,3};
```

即对应 $p_2(0)=1$，$p_3(0)=3$。

定义输入矢量如下。

```
P={3 4 5 6};
```

现在可以调用 sim 函数对上面所创建的自适应滤波网络进行仿真。

```
[A,Pf]=sim(net,P,Pi)
A =
    [54]    [79]    [94]    [118]
Pf =
    [5]    [6]
```

其中，Pf 为仿真后的延迟输入的终值。有兴趣的读者可以利用笔算对上述网络的仿真结构加以验证，以便能更好地理解自适应滤波网络的工作原理。

下面，利用 adapt 函数对自适应滤波网络进行训练，使自适应滤波网络能够根据输入序列 $\boldsymbol{P}$ 输出期望的目标序列 $\boldsymbol{T}$。

```
T={10 30 50 70};
net.adaptParam.passes=300;
[net,Y,E]=adapt(net,P,T,Pi)
```

经过训练，网络的实际输出响应为

```
Y =
    [14.2170]    [40.5601]    [43.9006]    [65.8289]
E =
    [-4.2170]    [-10.5601]    [6.0994]    [4.1711]
```

可见，训练结果基本满意，要想得到更高的匹配精度，还需要继续对网络进行训练。

此外，还可以采用 newlind 函数对上述自适应滤波网络直接进行设计，即

```
net=newlind(P,T,Pi);
Y=sim(net,P)
Y =
    [10.0000]    [30.0000]    [50.0000]    [70.0000]
```

可见，利用 newlind 函数直接设计出的自适应滤波网络实现了零误差输出。

4.2.4 线性神经网络的局限性

线性神经网络只能反映输入和输出样本矢量间的线性映射关系，和感知器神经网络一样，它也只能解决线性可分问题。由于线性神经网络的误差曲面是一个多维抛物面，所以在学习速率足够小的情况下，对于基于最小二乘梯度下降原理进行训练的线性神经网络总可以找到一个最优解。但是，尽管如此，对线性神经网络的训练并不一定总能达到零误差。线性神经网络的训练性能要受网络规则和训练样本集大小的限制。若线性神经网络的自由度（即神经网络所有权值和阈值的个数总和）小于训练样本集中“输入—目标”矢量的对数，且各样本矢量线性无关，则网络训练不可能达到零误差，而只能得到一个使网络误差最小的解；反之，若网络自由度大于样本集的个数，则会得到无穷多个使网络训练误差为零的解。此外，值得注意的是，线性神经网络的训练和性能都受到学习速率参数的影响，过大的学习速率可能会导致网络性能发散。

4.2.5 线性神经网络的 MATLAB 应用举例

【例 4-6】用直接设计法设计一个简单的线性神经元，使其能够拟合如下所给的输入样本矢量和目标矢量，其中

输入矢量为

```
P=[008 -2]
```

目标矢量为

```
T=[0.5 1]
```

本例的 MATLAB 代码如下。

```
clear all
```

```
echo on;
pause    %敲任意键开始
P=[0.8 -2];
T=[0.5 1];
%设定线性神经元可能的权值和阈值范围
W_range=-1:0.2:1;
B_range=-1:0.2:1;
%根据输入和目标矢量绘制线性神经元的误差曲面
ES=errsurf(P,T,W_range,B_range,'purelin');
plotes(W_range,B_range,ES);
%直接设法设计线性神经网络
net=newlind(P,T);
%对线性神经网络进行仿真
A=sim(net,P)
%计算仿真误差
E=T-A
SE=sse(E)
%根据网络权值和阈值在误差曲面上绘制当前位置点
plotes(W_range,B_range,ES);
plotep(net.iw{1,1},net.b{1},SE);
echo off;
```

窗口显示结果如下。

```
A =
    0.5000     1.0000
E =
     0     0
SE =
     0
```

在利用 newlind 函数设计线性神经元之前，可以根据可能的网络权值和阈值范围，并利用函数 errsurf 绘制出神经元的误差曲面。本例所绘制的误差曲面如图 4-21 所示，其中越亮的部分误差越小。图 4-21 中所示的亮点为网络设计完成之后根据当前网络权值和阈值，并利用 plotep 函数绘制的误差位置点，可见误差点位于误差曲面的最低点。

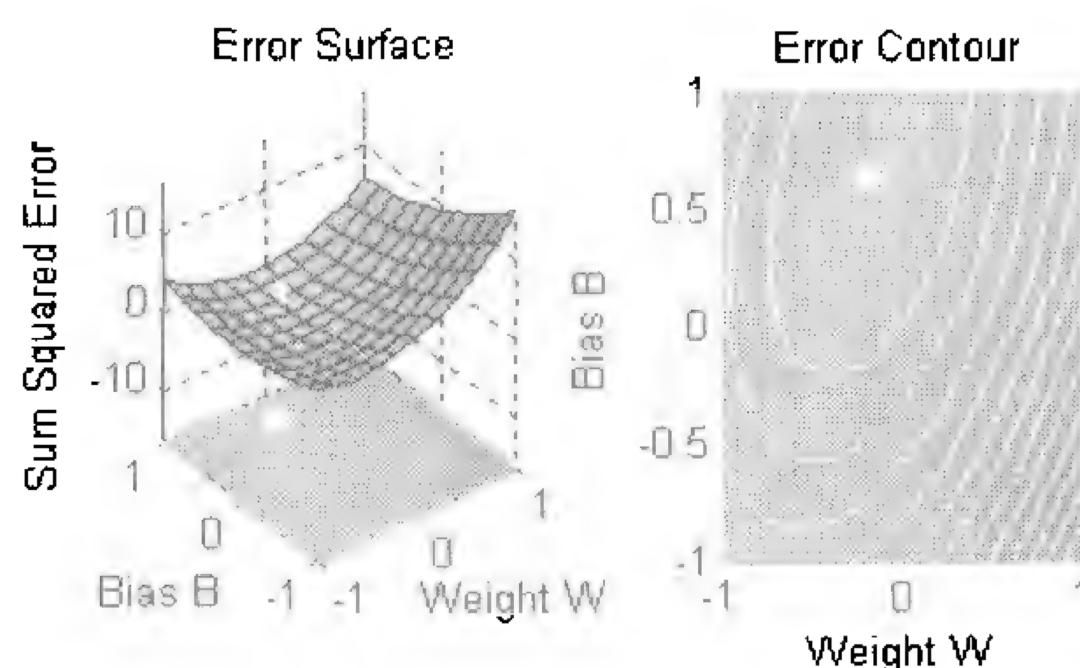


图 4-21　线性神经元的误差曲面和误差点

【例 4-7】设计并训练一个线性网络，实现从输入向量 $\boldsymbol{P}$ 到输出向量 $\boldsymbol{T}$ 的转换。

```
P=[1 2 4;2 4 8];
T=[0.5 1 -1];
```

从两个向量的结构来看，所设计的线性网络是具有阈值的，并且输入层的神经元为 2 个，

输出层的神经元为 1 个。如果仔细观察，就会发现输入向量 ***P*** 是线性相关的。在这种情况下，线性网络能够解决这个问题。其 MATLAB 代码如下。

```
P=[1 2 4;2 4 8];
T=[0.5 1 -1];
net=newlin(minmax(P),1,0,0.01);
Y=sim(net,P);
net=init(net);
net.trainParam.epochs=2000;
net.trainParam.goal=0.0001;
net=train(net,P,T);
```

网络训练结果为

```
TRAINB, Epoch 0/2000, MSE 0.75/0.0001.
TRAINB, Epoch 25/2000, MSE 0.576995/0.0001.
……
TRAINB, Epoch 1975/2000, MSE 0.214286/0.0001.
TRAINB, Epoch 2000/2000, MSE 0.214286/0.0001.
TRAINB, Maximum epoch reached.
```

可见，网络训练第 1000 步至 1500 步时，网络误差发生了轻微的改变，从 1500 步后，网络误差就不再改变，如图 4-22 所示，此时的网络训练误差为 0. 214286 左右。

对网络进行仿真，得到此时的网络输出 ***Y***。

```
Y=sim(net,P)
Y =
    0.9286    0.3571    -0.7857
```

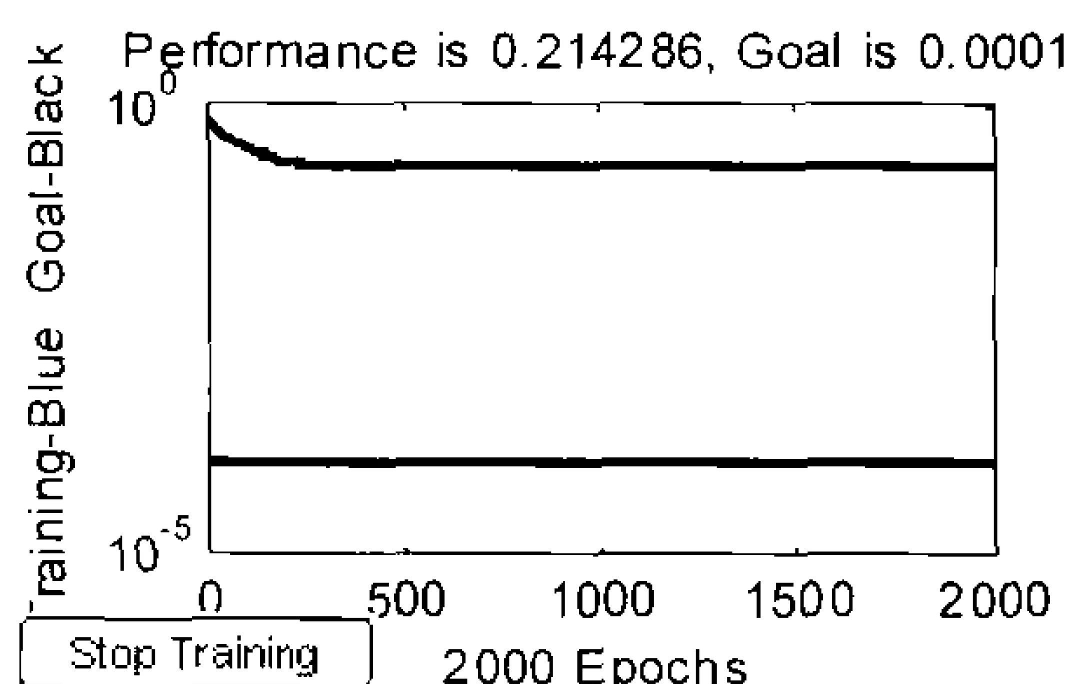


图 4-22　训练误差曲线

此时 ***Y*** 与目标向量 T 相比较，发现误差还是比较大的。这说明在这种情况下，网络是无法给出完美的解决方案的。

【例 4-8】利用线性神经网络求取非线性问题的最佳线性拟合解。

从前面已经知道，线性神经网络只能描述输入、输出间的线性映射关系，当输入、输出间是非线性关系时，利用线性神经网络只能得到输入、输出间的最佳线性拟合解，而不能实现完全的线性拟合。本例我们将采用线性单神经元对如下定义的输入和目标矢量进行拟合，即输入矢量为 ***P***=[1 1.2 3 -1.2]；目标矢量为 ***T***=[0.4 1 3 -0.5]。

由输入矢量和目标矢量可见，两者之间是非线性关系，即找不到一组权值 ***W*** 和阈值 ***B*** 使得对所有输入和目标样本矢量均满足线性表达式 ***T***=***W***×***P***+***B***。本例中，我们将采用 newlind 函数直接设计法和 train 函数设计法两种方案对线性神经网络进行设计。完整的 MATLAB 代码如下。

```
clear
clf reset
echo on
%定义输入目标和目标矢量
P=[1 1.2 3 -1.2];
T=[0.4 1 3 -0.5];
pause
%设定线性神经元可能的权值和阈值范围
W_range=-2:0.2:2;
B_range=-2:0.2:2;
%根据输入和目标矢量绘制线性神经元的误差曲面
ES=errsurf(P,T,W_range,B_range,'purelin');
plotes(W_range,B_range,ES);
pause
echo off
disp('1.采用 newlind 函数直接设计线性神经网络');
disp('2.采用 train 函数训练并设计线性神经网络');
choice=input('请选择设计方案（1，2）：');
if (choice==1)
    echo on
    %用 newlind 函数直接设计线性神经网络
    net=newlind(P,T);
    pause
    %对性线神经网络进行仿真
    A=sim(net,P)
    %计算仿真误差
    E=T-A
    SE=sse(E)
    pause
    %在误差曲面上绘制当前误差位置点
    plotep(net.iw{1,1},net.b{1},SE);
    pause
    echo off
    echo on
    %绘制拟合曲线
    plot(P,T,'r+');
    hold on;
    plot(P,A);
    pause
else
    echo on
    %用 train 函数训练并设计线性神经网络
    %计算最快的稳定学习速率值
    maxlr=maxlinlr(P,'bias')
    %创建线性神经网络
    net=newlin(minmax(P),1,[0],0.5*maxlr);
    %在误差曲面上选择权值和阈值对线性神经元进行初始化
```

```
        plotes(W_range,B_range,ES);
        subplot(1,2,2);
        [net.iw{1,1},net.b{1}]=ginput(1);
        echo off
        SE=sse(T-sim(net,P));
        plotep(ent.iw{1,1},net.b{1},SE);
        echo on
        %对线性神经网络进行训练
        net.trainParam.goal=0.01;
        [net,tr]=train(net,P,T);
        pause;
        %对线性神经网络进行仿真
        A=sim(net,P)
        E=T-A
        SE=sse(E)
        pause
        %在误差曲面上绘制当前误差位置点
        plotep(net.iw{1,1},net.b{1},SE);
        pause
        %绘制误差变化曲线
        plotperf(tr,net.trainParam.goal);
        pause
        %绘制拟合曲线
        plot(P,T,'b*');
        hold on;
        plot(P,A);
        pause
    end
    echo off
```

按任意键执行代码，命令窗口显示如下内容给用户选择设计方式。

```
（1）采用 newlind 函数直接设计线性神经网络。
（2）采用 train 函数训练并设计线性神经网络。
请选择设计方案
（1，2）：1
```

在此选择方案 1 后按回车键显示结果为

```
A =
     0.9750      1.1394      2.6191     -0.8336
E =
    -0.5750     -0.1394      0.3809      0.3336
SE =
     0.6064
```

实践证明，两种方案均未能达到拟合零误差的效果，但两种方案都得到了相同的最佳拟合结果，图 4-23 所示为拟合曲线。此外，图 4-24 给出了所设计的线性神经元的误差曲面和误差点位置。

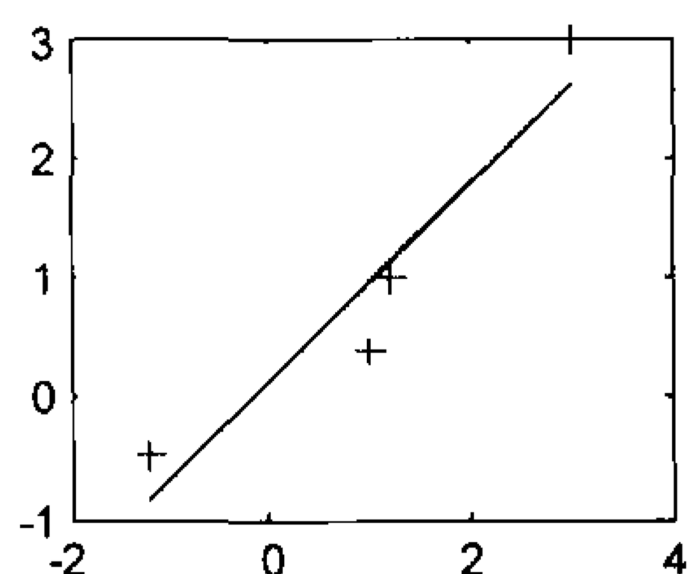


图 4-23　线性神经元最佳拟合曲线

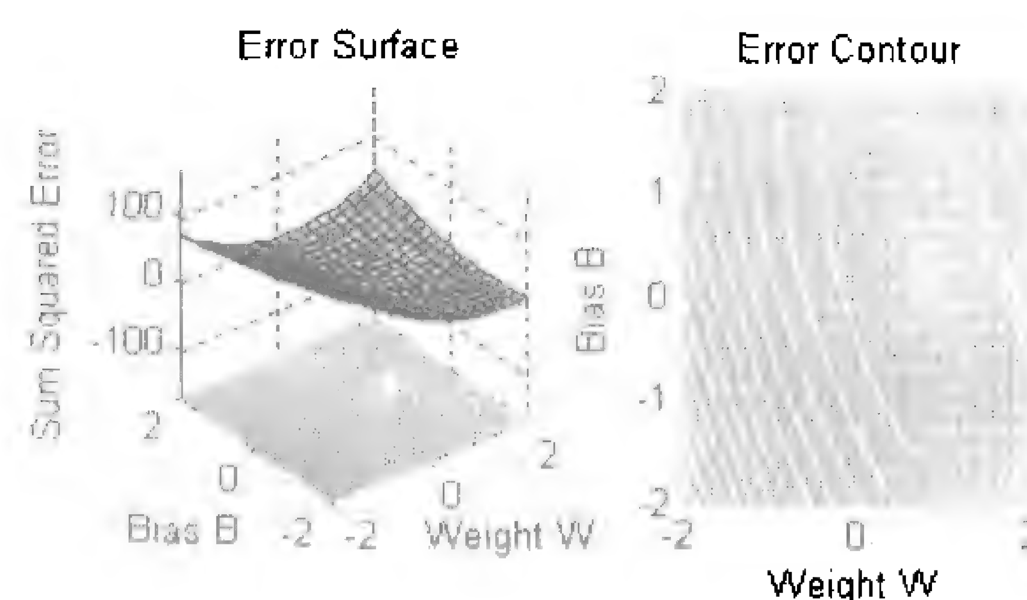


图 4-24　线性神经元的误差曲面和误差点

4.3　BP 网络

感知器神经网络的学习规则只能训练单层神经网络，而单层神经网络只能解决线性可分的分类问题。多层神经网络可以用于非线性分类问题，但需要寻找训练多层网络的学习算法。

1974 年 P.Werbos 在其博士论文中提出了第一个适合多层网络的学习算法，但该算法并未受到足够的重视和广泛的应用，直到 20 世纪 80 年代中期，美国加利福尼的 PDP（Parallel Distributed procession）小组于 1986 年发表了 Parallel Distributed Processing 一书，将该算法应用于神经网络的研究，才使之成为迄今为止最著名的多层网络学习算法——BP 算法，由此算法训练的神经网络，称为 BP 神经网络。在人工神经网络的实际应用中，BP 网络广泛应用于函数逼近、模式识别/分类、数据压缩等，80%～90%的人工神经网络模型是采用 BP 网络或它的变化形式，它也是前馈网络的核心部分，体现了人工神经网络最精华的部分。

4.3.1　BP 神经元及其模型

BP 神经元模型如图 4-25 所示。

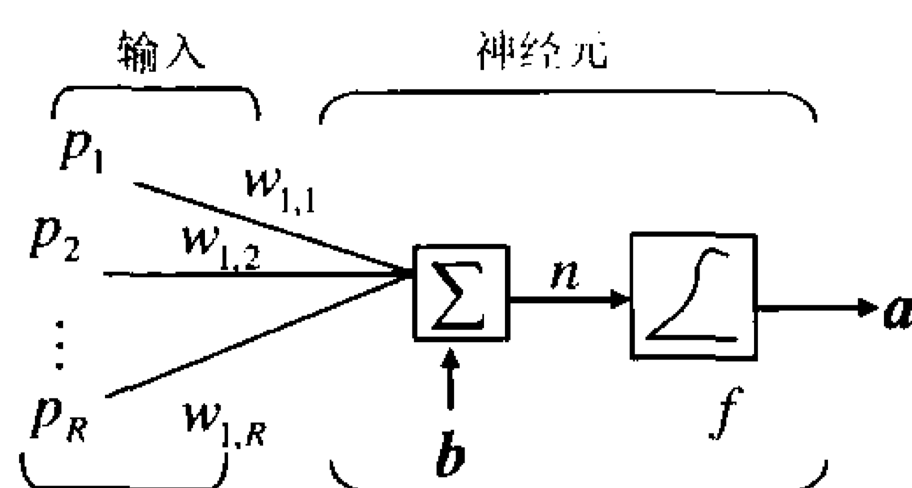


图 4-25　BP 神经元模型

BP 神经元与其他神经元类似，不同的是 BP 神经元的传输函数为非线性函数，最常用的函数是 logsig 和 tansig 函数，有的输出层也采用线性函数（purelin）。其输出为

$$\boldsymbol{a} = \text{logsig}(\boldsymbol{Wp} + \boldsymbol{b}) \tag{4-21}$$

BP 网络一般为多层神经网络。由 BP 神经元构成的两层网络如图 4-26 所示。BP 网络的信息从输入层流向输出层，因此是一种多层前馈神经网络。

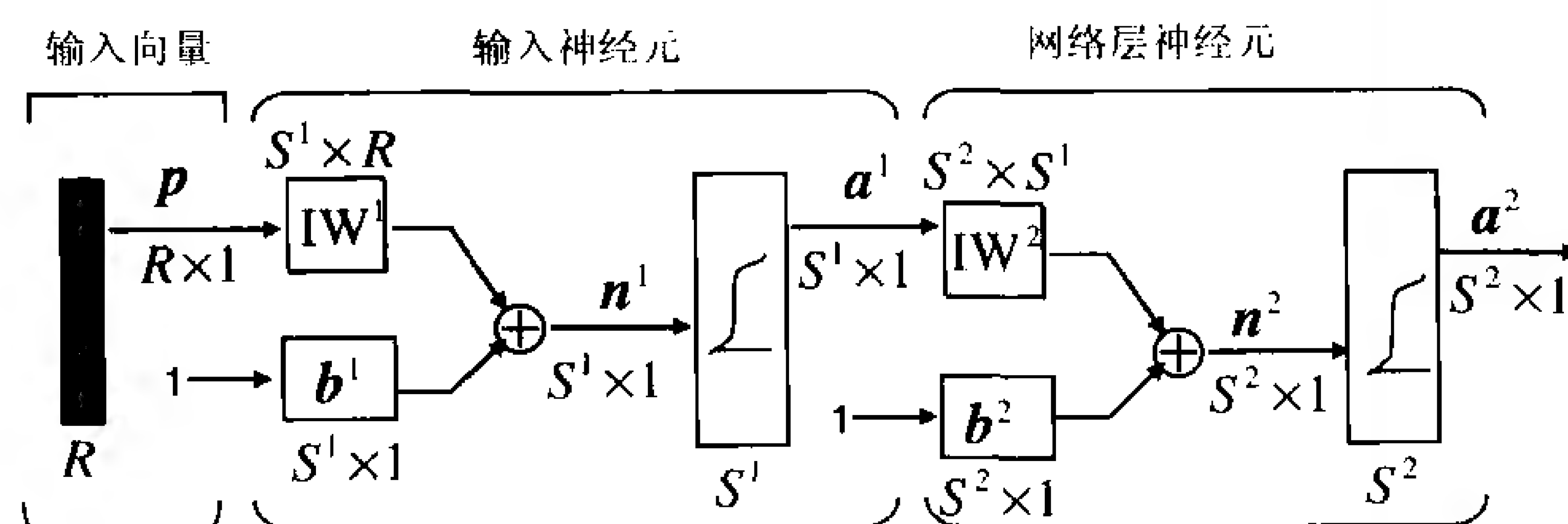


图 4-26　两层 BP 神经网络模型

如果多层 BP 网络的输出层采用 S 型传输函数（如 logsig），其输出值将会限制在一个较小的范围内 [0,1]；而采用线性传输函数则可以取任意值。

4.3.2 BP 网络的学习

在确定了 BP 网络的结构后，要通过输入和输出样本集对网络进行训练，亦即对网络的阈值和权值进行学习和修正，以使网络实现给定的输入输出映射关系。

BP 网络的学习过程分为两个阶段：

第一个阶段是输入已知学习样本，通过设置的网络结构和前一次迭代的权值和阈值，从网络第一层向后计算各神经元的输出。

第二个阶段是对权值和阈值进行修改，从最后一层向前计算各权值和阈值对总误差的影响（梯度），据此对各权值和阈值进行修改。

以上两个过程反复交替，直到达到收敛为止。由于误差逐层往回传递，以修正层与层间的权值和阈值，所以称该算法为误差反向传播（back propagation）算法，这种误差反传学习算法可以推广到有若干个中间层的多层网络，因此该多层网络常称之为 BP 网络。标准的 BP 算法和 Widrow-Hoff 学习规则一样，也是一种梯度下降学习算法，其权值的修正是沿着误差性能函数梯度的反方向进行的。针对标准 BP 算法存在的一些不足，出现了几种基于标准 BP 算法的改进算法，如变梯度算法、牛顿算法等。

1．BP 网络学习算法

1）最速下降 BP 算法（Steepest Descent Backpropagation，SDBP）

（1）最速下降 BP 算法。

对于图 4-26 所示的 BP 神经网络，设 k 为迭代次数，则每一层权值和阈值的修正按下式进行

$$\boldsymbol{x}(k+1)=\boldsymbol{x}(k)-\alpha\boldsymbol{g}(k) \tag{4-22}$$

式中，$\boldsymbol{x}(k)$ 为第 k 次迭代各层之间的连接权向量或阈值向量。

$\boldsymbol{g}(k)=\dfrac{\partial \boldsymbol{E}(k)}{\partial \boldsymbol{x}(k)}$ 为第 k 次迭代的神经网络输出误差对各权值或阈值的梯度向量。负号表示梯度的反方向，即梯度的最速下降方向。

α 为学习速率，在训练时是一常数。在 MATLAB 神经网络工具箱中，其默认值为 0.01，可以通过改变训练参数进行设置。

$\boldsymbol{E}(k)$ 为第 k 次迭代的网络输出的总误差性能函数，在 MATLAB 神经网络工具箱中，BP 网络误差性能函数默认值为均方误差 MSE（Mean Square Error），以两层 BP 网络为例，只有一个输入样本时，有

$$\boldsymbol{E}(k)=\boldsymbol{E}[e^2(k)]\approx\frac{1}{S^2}\sum_{i=1}^{S^2}[t_i^2-a_i^2(k)]^2 \tag{4-23}$$

$$a_i^2(k)=f^2\left\{\sum_{j=1}^{S^2}\left[w_{i,j}^2(k)a_i^1(k)-b_i^2(k)\right]\right\}$$
$$=f^2\left\{\sum_{j=1}^{S^2}\left[w_{i,j}^2(k)f^1\left(\sum_{j=1}^{S^1}(iw_{i,j}^1(k)p_i+ib_i^1(k))\right)+b_i^2(k)\right]\right\} \tag{4-24}$$

若有 n 个输入样本

$$\boldsymbol{E}(k)=\boldsymbol{E}[e^2(k)]\approx\frac{1}{nS^2}\sum_{j=1}^{S^1}\sum_{i=1}^{S^2}[t_i^2-a_i^2(k)]^2 \tag{4-25}$$

根据式（4-23）或式（4-25）和各层的传输函数，可以求出第 k 次迭代的总误差曲面的梯度 $\boldsymbol{g}(k)=\dfrac{\partial\boldsymbol{E}(k)}{\partial\boldsymbol{x}(k)}$，分别代入式（4-22），便可以逐次修正其权值和阈值，并使总的误差向减小的方向变化，直到达到所要求的误差性能为止。

从上述过程可以看出，权值和阈值的修正是在所有样本输入后，计算其总的误差后进行的，这种修正方式称为批处理。在样本数比较多时，批处理方式比分别处理方式的收敛速度快。

注意

在 MATLAB 神经网络工具箱中，采用最速下降 BP 算法中的训练函数为 traingd。

采用 traingd 训练 BP 网络的方法是：

① 将网络训练函数（trainFcn）设置为 traingd，每个神经网络只有一个训练函数与之对应；

② 然后调用训练函数 train；

③ 与 traingd 相关的训练参数有 7 个：epochs、show、goal、time、min_grad、max_fail 和 lr。训练函数 traingd 的说明参见第 3 章。

（2）最速下降 BP 算法的误差曲面。

对于图 4-26 所示的 BP 神经网络，权值空间的维数为

$$n_w=R\times S^1+S^2\times S^1 \tag{4-26}$$

阈值空间的维数为

$$n_b=S^1+S^2 \tag{4-27}$$

根据式（4-23）至式（4-27）可以看出，若要同时调整所有的权值和阈值，则误差函数的空间维数为

$$n_E=n_w+n_b=R\times S^1+S^2\times S^1+S^1+S^2 \tag{4-28}$$

一般 $n_E>2$，所以误差曲面是一个具有复杂形状的 n_E 维超曲面，无法在三维空间表示出来，当然，可以只让其中的二维变量改变，而固定其他维变量，画出关于该二维变量的误差曲面图。对于单输入单个神经元，在 MATLAB 中其误差曲面可以由函数 errsurf 直接绘制，以图 4-27 所示的单输入单个 BP 神经元为例，设训练样本集为

P=[-6.0 -6.1 -4.1 -4.0 4.0 4.1 6.0 6.1];

T=[0 0.97 0.99 0.01 0.03 1 1];

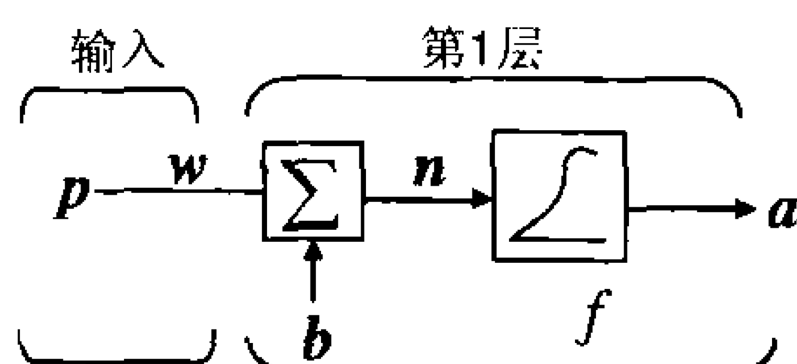


图 4-27　单输入单个 BP 神经元

则绘制其误差曲面的 MATLAB 程序代码如下。

```
P=[-6.0 -6.1 -4.1 -4.0 4.0 4.1 6.0 6.1];
T=[0 0 0.97 0.99 0.01 0.03 1 1];
W=-1:0.1:1;
B=-2.5:0.25:2.5;
ES=errsurf(P,T,W,B,'logsig');
plotes(W,B,ES,[60 30]);
```

其误差曲面是关于 **W** 和 **B** 的二维曲面，如图 4-28 所示。

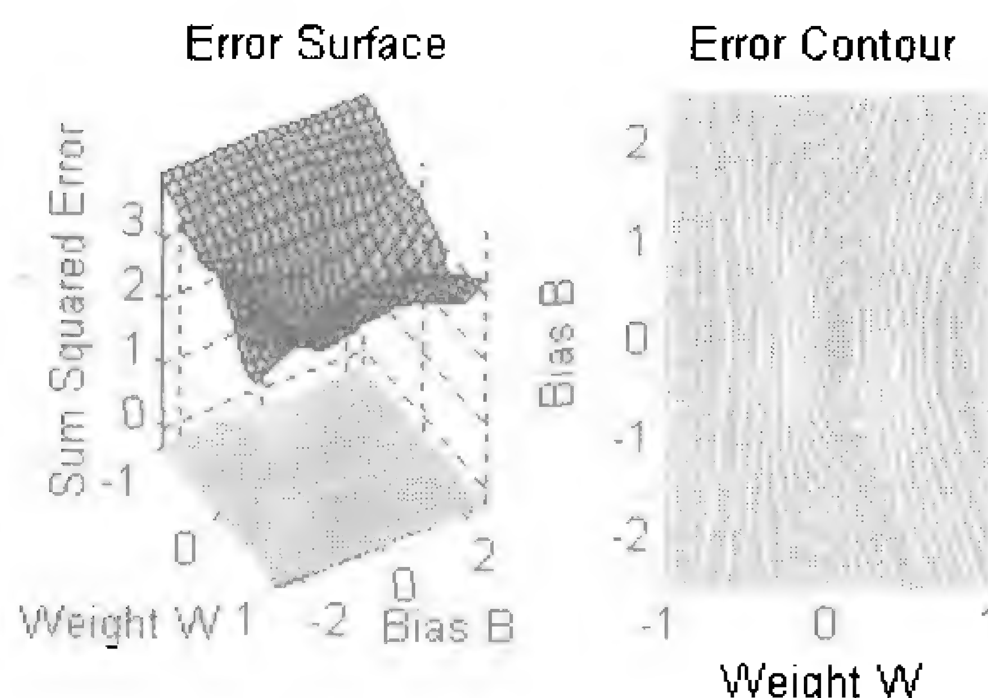


图 4-28　单个 BP 神经元的误差曲面及其等值线

因为 BP 神经元的传输函数为非线性函数，所以其误差函数往往有多个极小点。若误差曲面有两个极小点 m 和 n，则当学习过程中，如果误差先到达局部极小点 m 点，在该点的梯度为 $g_k=0$，则按式（4-22）将无法继续调整权值和阈值，学习过程结束，但尚未到达全局极小点 n。

另外一种情况是学习过程发生振荡，误差曲面在 m 点和 n 点的梯度大小相同，但方面相反，如果第 k 次学习使误差落在 m 点，而第 $k+1$ 次学习又恰好使误差落在 n 点，那么按式（4-22）进行的权值和阈值调整，将在 m 点和 n 点重复进行，从而形成振荡。

从以上分析可以看出，最速下降 BP 算法可以使权值和阈值向量得到一个稳定的解，但存在一些缺点，如收敛速度慢、网络易陷于局部极小、学习过程常常发生振荡等。为了克服其不足，出现了许多改进算法。

2）动量 BP 算法（Momentum Backpropagation，MOBP）

动量 BP 算法是在梯度下降算法的基础上引入动量因子 $\eta(0<\eta<1)$。

$$\Delta \boldsymbol{x}(k+1)=\eta\Delta \boldsymbol{x}(k)+\alpha(1-\eta)\frac{\partial \boldsymbol{E}(k)}{\partial \boldsymbol{x}(k)} \tag{4-29}$$

$$\boldsymbol{x}(k+1)=\boldsymbol{x}(k)+\Delta \boldsymbol{x}(k+1) \tag{4-30}$$

该算法是以前一次的修正结果来影响本次修正量，当前一次的修正量过大时，式（4-29）第 2 项的符号将与前一次修正量的符号相反，从而使本次的修正量减小，达到减小振荡的作用；当前一次的修正量过小时，式（4-29）第 2 项的符号将与前一次修正量的符号相同，从而使本次的修正量增大，达到加速修正的作用。可以看出，动量 BP 算法，总是力图使在同一梯度方向上的修正量增加。动量因子 η 越大，同一梯度方向上的“动量”也越大。

在动量 BP 算法中，可以采用较大的学习率，而不会造成学习过程的发散，因为当修正过量时，该算法（即动量 BP 算法）总是可以使修正量减小，以保持修正方向向着收敛的方向进行；另一方面，动量 BP 算法总是加速同一梯度方向的修正量。上述两个方面表明，在保证算法稳定的同时，动量 BP 算法的收敛速率较快，学习时间较短。

注意

在 MATLAB 神经网络工具箱中，采用动量 BP 算法的训练函数为 traingdm。

函数 traingdm 与函数 traingd 一样，它不是被用户直接调用的，而是通过训练函数 train 调用的，所以在使用 traingdm 时必须将网络训练函数（trainFcn）设置为 traingdm。

3）学习率可变的 BP 算法（Variable Learning Rate Backpropagation，VLBP）

在最速下降 BP 算法和动量 BP 算法中，其学习率是一个常数，在整个训练过程中保持不变，学习算法的性能对于学习率的选择非常敏感，学习率过大，算法可能振荡而不稳定；学习率过小，则收敛速度慢，训练时间长。而在训练之前，要选择最佳的学习率是不现实的。事实上，可以在训练的过程中，使学习率随之变化，而使算法沿着误差性能曲面进行修正案。

自适应调整学习率的梯度下降算法，在训练的过程中，力图使算法稳定，而同时又使学习的步长尽量得大，学习率则是根据局部误差曲面作出相应的调整。当误差以减小的方式趋于目标时，说明修正方向正确，可使步长增加，因此学习率乘以增量因子 k_{inc}，使学习率增加；而当误差增加超过事先设定的值时，说明修正过头，应减小步长，因此学习率乘以减量因子 k_{dec}，使学习率减小，同时舍去使误差增加的前一步修正过程，即

$$a(k+1)=\begin{cases} k_{\text{inc}}a(k) & \boldsymbol{E}(k+1)<\boldsymbol{E}(k) \\ k_{\text{dec}}a(k) & \boldsymbol{E}(k+1)>\boldsymbol{E}(k) \end{cases} \tag{4-31}$$

注意

在 MATLAB 神经网络工具箱中，采用学习率可变的最速下降 BP 算法的训练函数为 traingda; 采用学习率可变的动量 BP 算法的训练函数为 traingdx。

4）弹性算法（Resilient Back-PROPagation，RPROP）

多层 BP 网络的隐层一般采用传输函数 sigmoid，它把一个取值范围为无穷大的输入变量，压缩到一个取值范围有限的输出变量中。函数 sigmoid 具有这样的特性：当输入变量的取值很大时，其斜率趋于零，这样在采用最速下降 BP 法训练传输函数为 sigmoid 的多层网络时就带来一个问题，尽管权值和阈值离其最佳值相差甚远，但此时梯度的幅度非常小，导致权值和阈值的修正量也很小，这样就使训练的时间变得很长。

RPROP 算法的目的是消除梯度幅度的不利影响，所以在进行权值的修正时，仅仅用到偏导的符号，而其幅值却不影响权值的修正，权值大小的改变取决于与幅值无关的修正值。当连续两次迭代的梯度方向相同时，可将权值和阈值的修正值乘以一个增量因子，使其修正值增加；当连续两次迭代的梯度方向相反时，可将权值和阈值的修正值乘以一个减量因子，使其修正值减小；当梯度为零时，权值和阈值的修正值保持不变；当权值的修正发生振荡时，其修正值将会减小。如果权值在相同的梯度上连续被修正，则其幅度必将增加，从而克服了梯度幅度偏导的不利影响，即

$$\Delta x(k+1)=\Delta x(k+1)\cdot \mathrm{sign}(g(k))$$

$$=\begin{cases}\Delta x(k)\cdot k_{\mathrm{inc}}\cdot \mathrm{sign}(g(k)) & \text{（当连续两次迭代的梯度方向相同时）}\\ \Delta x(k)\cdot k_{\mathrm{dec}}\cdot \mathrm{sign}(g(k)) & \text{（当连续两次迭代的梯度方向相反时）}\\ \Delta x(k) & \text{（当 } g(k)=0 \text{ 时）}\end{cases} \tag{4-32}$$

式中，$g(k)$ 为第 k 次迭代的梯度；$\Delta x(k)$ 为权值或阈值第 k 次迭代的幅度修正值，其初始值 $\Delta x(0)$ 是用户设置的；增量因子 k_{inc} 和减量因子 k_{dec} 也是用户设置的。

注意

在 MATLAB 神经网络工具箱中，RPROP 算法的训练函数为 trainrp

5）变梯度算法（Conjugate Gradient Backpropagation，CGBP）

最速下降 BP 算法是沿着梯度最陡下降方向修正权值的，虽然误差函数沿着梯度的最陡下降方向进行修正，误差减小的速度是最快的，但收敛的速度不一定是最快的。在变梯度算法中，沿着变化的方向进行搜索，使其收敛速度比最陡下降梯度方向的收敛速度更快。

所有变梯度算法的第一次迭代都是沿着最陡梯度下降方向开始进行搜索的

$$p(0)=-g(0) \tag{4-33}$$

然后，决定最佳距离的线性搜索沿着当前搜索的方向进行

$$x(k+1)=x(k)+\alpha\, p(k) \tag{4-34}$$

$$p(k)=-g(k)+\beta(k)p(k-1) \tag{4-35}$$

式中，$p(k)$ 为第 $k+1$ 次迭代的搜索方向，从式（4-35）可以看出，它由第 k 次迭代的梯度和搜索方向共同决定；系数 $\beta(k)$ 在不同的变梯度算法中有不同的计算方法。

（1）Fletcher-Reeves 修正算法。

Fletcher-Reeves 修正算法是由 R.Fletcher 和 C.M.Reeves 提出的，在式（4-35）中，系数 $\beta(k)$ 定义为

$$\beta(k)=\frac{g^{\mathrm{T}}(k)g(k)}{g^{\mathrm{T}}(k-1)g(k-1)} \tag{4-36}$$

这种变梯度算法的速度通常比变学习率算法的速率快得多，有时比 RPROP 算法还快。其所需的存储空间也只比普通算法略多一点，所以在连接权的数量很多时，时常选用该算法。

注意

在 MATLAB 神经网络工具箱中，采用 Fletcher-Reeves 修正算法的训练函数为 traincgf。

（2）Polak-Ribiere 修正算法。

Polak-Ribiere 算法是由 Polak 和 Ribiere 提出的，在式（4-35）中，系数 $\beta(k)$ 定义为

$$\beta(k)=\frac{\Delta \boldsymbol{g}^{\mathrm{T}}(k-1)\boldsymbol{g}(k)}{\boldsymbol{g}^{\mathrm{T}}(k-1)\boldsymbol{g}(k-1)} \tag{4-37}$$

此时 Fletcher-Reeves 修正算法就演变成 Polak-Ribiere 修正算法。

Polak-Ribiere 修正算法的性能与 Fletcher-Reeves 修正算法相差无几，但存储空间比 Fletcher-Reeves 修正算法略大。

注意

在 MATLAB 神经网络工具箱中，采用 Polak-Ribiere 修正算法的训练函数为 traincgp。

（3）Powell-Beale 复位算法。

对于所有的变梯度算法，搜索方向都会周期性地被复位成负的梯度方向，通常复位点出现在迭代次数和网络参数个数（权值和阈值）相等的地方，为了提高训练的有效性，另外一些复位的算法被提出，其中 Powell-Beale 复位算法是由 Beale 和 Powell 首先提出的。在此算法中，如果梯度满足下式

$$\left|\boldsymbol{g}^{\mathrm{T}}(k-1)\boldsymbol{g}(k)\right| \geqslant 0.2\left\|\boldsymbol{g}(k)\right\|^{2} \tag{4-38}$$

则搜索方向被复位成负的梯度方向，即 $\boldsymbol{p}(k)=-\boldsymbol{g}(k)$ 。

尽管对于任意给定的一个问题，该算法的性能难以预先确定，但可以肯定，在处理某些问题上 Powell-Beale 复位算法的性能比 Polak-Ribiere 修正算法要略好些，其存储空间则比 Polak-Ribiere 修正算法要略大些。

注意

在 MATLAB 神经网络工具箱中，采用 Powell-Beale 修正算法的训练函数为 traincgb。

（4）SCG（Scaled Conjugate Gradient）算法。

到目前为止我们讨论的各种变梯度算法在每次迭代时都需要确定线性搜索方向，而线性搜索的计算需要付出的代价是很大的，因为每一次搜索都需要对全部训练样本的网络响应进行多次计算。SCG 算法是由 Moller 提出的改进算法，它不需要在每一次迭代中都进行线性搜索，从而避免了搜索方向计算的耗时问题。其基本思想采用了模型信任区间逼近的原理。

SCG 算法也许比其他变梯度算法需要更多的迭代次数，但由于不需要在迭代中进行线性搜索，所以每次迭代的计算量大大减小。SCG 算法所需要的存储空间与 Fletcher-Reeves 修正算法的存储空间相差无几。

6）拟牛顿算法（Quasi-Newton algorithms）

牛顿法是一种基于二阶泰勒（Taylor）级数的快速优化算法。其基本方法是

$$\boldsymbol{x}(k+1)=\boldsymbol{x}(k)-\boldsymbol{A}^{-1}(k)\boldsymbol{g}(k) \tag{4-39}$$

式中，$A(k)$ 为误差性能函数在当前权值和阈值下的 Hessian 矩阵（二阶导数）

$$\boldsymbol{A}(k)=\nabla^{2}\boldsymbol{F}(x)\big|_{\boldsymbol{x}=\boldsymbol{x}(k)} \tag{4-40}$$

牛顿法通常比变梯度法的收敛速度快，但前馈神经网络计算 Hessian 矩阵是很复杂的，付出的代价也很大。

有一类基于牛顿法的算法不需要求二导数，此类方法称为拟牛顿法（或正切法），在算法中的 Hessian 矩阵用其近似值进行修正，修正值被看成梯度的函数。

（1）BFGS（Boryden、Fletcher、Goldfarb、Shanno）算法。

在公开发表的研究成果中，拟牛顿法应用最为成功的有 Boryden、Fletcher、Goldfarb 和 Shanno 修正算法，合称为 BFGS 算法。

该算法虽然收敛所需的步长通常较少，但在每次迭代过程中所需要的计算量和存储空间比变梯度算法都要大，对近似 Hessian 矩阵必须进行存储，其大小为 $n \times n$，这里 n 为网络的连接权和阈值的数量。所以对于规模很大的网络用 RPROP 算法或任何一种变梯度算法可能好些；而对于规模较小的网络则用 BFGS 算法可能更有效。



在 MATLAB 神经网络工具箱中，采用 BFGS 算法的训练函数为 trainbfg。

（2）OSS（One Step Secant）算法。

由于 BFGS 算法在每次迭代时比变梯度算法需要更多的存储空间和计算量，所以对于正切近似法减少其存储量和计算量是必要的。OSS 算法试图解决变梯度法和拟牛顿（正切）法之间的矛盾，该算法不必存储全部 Hessian 矩阵，它假定每一次迭代时，前一次迭代的 Hessian 矩阵具有一致性，这样作的另外一个优点是，在新的搜索方向进行计算时不必计算矩阵的逆。

该算法每次迭代所需的存储量和计算量介于梯度算法和完全拟牛顿算法之间。



在 MATLAB 神经网络工具箱中，采用 OSS 算法的训练函数为 trainoss。

7）LM（Levenberg-Marquardt）算法

LM 算法与拟牛顿法一样，是为了在以近似二阶训练速率进行修正时避免计算 Hessian 矩阵而设计的。当误差性能函数具有平方和误差（训练前馈网络的典型误差函数）的形式时，Hessian 矩阵可以近似表示为

$$\boldsymbol{H} = \boldsymbol{J}^{\mathrm{T}}\boldsymbol{J} \tag{4-41}$$

梯度的计算表达式为

$$\boldsymbol{g} = \boldsymbol{J}^{\mathrm{T}}\boldsymbol{e} \tag{4-42}$$

式中，$\boldsymbol{H}$ 是包含网络误差函数对权值和阈值一阶导数的雅可比矩阵，$\boldsymbol{e}$ 是网络的误差向量。雅可比矩阵可以通过标准的前馈网络技术进行计算，比 Hessian 矩阵的计算要简单得多。

类似于牛顿法，LM 算法用上述近似 Hessian 矩阵按下式进行修正。

$$\boldsymbol{x}(k+1) = \boldsymbol{x}(k) - [\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J} + \mu\boldsymbol{J}]^{-1}\boldsymbol{J}^{\mathrm{T}}\boldsymbol{e} \tag{4-43}$$

当系数 μ 为 0 时，上式即为牛顿法；当系数 μ 的值很大时，上式变为步长较小的梯度下降法。牛顿法逼近最小误差的速度更快、更精确，因此应尽可能使算法接近于牛顿法，在每一步成功

的迭代后（误差性能减小），使 μ 减小；仅在进行尝试性迭代后的误差性能增加的情况下，才使 μ 增加。这样，该算法每一步迭代的误差性能总是减小的。

LM 算法是为了训练中等规模的前馈神经网络（多达数百个连接权）而提出的最快速算法，它对 MATLAB 实现也是相当有效的，因为其矩阵的计算在 MATLAB 中是以函数实现的，其属性在设置时变得非常明确。



在 MATLAB 神经网络工具箱中，采用 LM 算法的训练函数为 trainlm。

2．BP 网络学习算法的比较

对于一个给定的问题，到底采用哪种训练方式，其训练速度最快，这里是很难预知的，因为这取决于许多因素，包括给定问题的复杂性、训练样本集的数量、网络权值和阈值的数量、误差目标、网络的用途（如用于模式识别还是函数逼近）等。

但通过实验，可以得出各种算法性能上的一些结论，通常对于包含数百个权值的函数逼近网络，LM 算法的收敛速度最快。如果要求的精度比较高，则该算法的优点尤其突出。在许多情况下，采用 LM 算法的训练函数 trainlm 可以获得比其他任何一种算法更小的均方误差。但当网络权值的数量增加时，trainlm 的优点将逐渐变得不很明显。另外，trainlm 对于模式识别相关问题的处理功能很弱，其存储空间比其他算法大，通过调整 trainlm 的存储空间参数 mem_reduc，虽然可以在一定程度上减小对存储空间的要求，但却需要增加运行时间。

将 RPROP 算法的训练函数 trainrp 应用于模式识别时，其速度是最快的，但对于函数逼近问题该算法却不是最好的，其性能同样会随着目标误差的减小而变差。该算法所需的存储空间较其他算法相对要小一些。

变梯度算法，特别是 SCG 算法，在更广泛的问题中，尤其是在网络规模较大的场合，其性能都很好。SCG 算法应用于函数逼近问题时，几乎与 LM 算法一样快（在网络规模较大时比 LM 算法更快）；而应用于模式识别时几乎与 RPROP 算法一样快，其性能不像 RPROP 算法随着目标误差的减小而下降得那么快。变梯度算法对存储空间的要求相对也低一些。

BFGS 算法类似于 LM 算法，其所需的存储空间比 LM 算法小，但其运算量却随网络的大小成几何级数增长，因为对每次迭代过程都必须计算相应矩阵的逆矩阵。

变学习率算法通常比其他算法的速度要慢很多，而其存储空间与 RPROP 算法一样，但在应用于某些问题时该算法仍然很有用。在有些特定的情形下收敛速度慢一些反而好些，例如，如果用收敛速度太快的算法，可能得到的结果是还达不到所要求的目标时训练就提前结束了，会错过使误差最小的点。

4.3.3　BP 网络的局限性

在人工神经网络的应用中，绝大部分的神经网络模型采用了 BP 网络及其变化形式，但这并不是说明 BP 网络是尽善尽美的，其各种算法依然存在一定的局限性。BP 网络的局限性主要有以下几个方面。

1）学习率与稳定性的矛盾

梯度算法进行稳定学习要求的学习率较小，所以通常学习过程的收敛速度很慢。附加动量法通常比较简单的梯度算法快，因为在保证稳定学习的同时，它可以采用很高的学习率，但对于许多实际应用，仍然太慢。以上两种方法往往只适用于希望增加训练次数的情况。如果有足

够的存储空间，则对于中、小规模的神经网络通常可采用 Levenberg-Mrquardt 算法；如果存储空间有问题，则可采用其他多种快速算法，例如对于大规模神经网络采用 trainscg 或 trainrp 更合适。

2）学习率的选择缺乏有效的方法

对于非线性网络，选择学习率也是一个比较困难的事情。对于线性网络，我们知道，学习率选择得太大，容易导致学习不稳定；反之，学习率选择得太小，则可能导致无法忍受的过长学习时间。不同于线性网络，我们还没有找到一个简单易行的方法，以解决非线性网络选择学习率的问题。对于快速训练算法，其默认参数值通常留有裕量。

3）训练过程可能陷于局部最小

从理论上说，多层 BP 网络可以实现任意可实现的线性和非线性函数的映射，克服了感知器和线性神经网络的局限性。但在实际应用中，BP 网络往往在训练过程中，也可能找不到某个具体问题的解，比如在训练过程中陷入局部最小的情况。当 BP 网络在训练过程中陷入误差性能函数的局部最小时，可以通过改变其初始值，并经过多次训练，以获得全局最小。

4）没有确定隐层神经元数的有效方法

确定多层神经网络隐层的神经元数也是一个很重要的问题，太少的隐层神经元会导致网络“欠适配”，太多的隐层神经元又会导致“过适配”。

4.3.4 BP 网络的 MATLAB 程序应用举例

1．BP 网络设计的基本方法

BP 网络的设计主要包括输入层、隐层、输出层及各层之间的传递函数几个方面。

1）网络层数

大多数通用的神经网络都预先确定了网络的层数，而 BP 网络可以包含不同的隐层。但理论上已经证明，在不限制隐层节点数的情况下，两层（只有一个隐层）的 BP 网络可以实现任意非线性映射。在模式样本相对较少的情况下，较少的隐层节点，可以实现模式样本空间的超平面划分，此时，选择两层 BP 网络就可以了；当模式样本数很多时，减小网络规模，增加一个隐层是必要的，但 BP 网络隐层数一般不超过两层。

2）输入层的节点数

输入层起缓冲存储器的作用，它接收外部的输入数据，因此其节点数取决于输入矢量的维数。比如，当把 32×32 大小的图像的像素作为输入数据时，输入节点数将为 1024。

3）输出层的节点数

输出层的节点数取决于两个方面，输出数据类型和表示该类型所需的数据大小。当 BP 网络用于模式分类时，以二进制形式来表示不同模式的输出结果，则输出层的节点数可根据待分类模式来确定。若设待分类模式的总数为 m，则有两种方法确定输出层的节点数。

① 节点数即为待分类模式总数 m，此时对应第 j 个待分类模式的输出为

$$O_j = \frac{[00\cdots010\cdots00]}{j}$$

即第 j 个节点输出为 1，其余输出均为 0。而以输出全为 0 表示拒识，即所输入的模式不属于待分类模式中的任何一种模式。

② 节点数为 $\log_2^m$ 个。这种方式的输出是 m 种输出模式的二进制编码。

4）隐层的节点数

一个具有无限隐层节点的两层 BP 网络可以实现任意从输入到输出的非线性映射。但对于有限个输入模式到输出模式的映射，并不需要无限个隐层节点，这就涉及如何选择隐层节点数的问题，而这一问题的复杂性，使得至今为止，尚未找到一个很好的解析式，隐层节点数往往根据前人设计所得的经验和自己进行试验来确定。一般认为，隐层节点数与求解问题的要求、输入输出单元数多少都有直接的关系。另外，隐层节点数太多会导致学习时间过长；而隐层节点数太少，则容错性差，识别未经学习样本的能力低，所以必须综合多方面的因素进行设计。

对于用于模式识别/分类的 BP 网络，根据前人经验，可以参照以下公式进行设计

$$n=\sqrt{n_\mathrm{i}+n_\mathrm{o}}+a \tag{4-44}$$

式中，n 为隐层节点数；n_i 为输入节点数；n_o 为输出节点数；a 为 1～10 之间的常数。

5）传输函数

BP 网络中的传输函数通常采用 S（sigmoid）型函数

$$f=\frac{1}{1+e^{-x}} \tag{4-45}$$

在某些特定的情况下，还可能采用纯线性（pureline）函数。如果 BP 网络的最后一层是 sigmoid 函数，那么整个网络的输出就限制在一个较小的范围内（0～1 之间的连续量）；如果 BP 网络的最后一层是 pureline 函数，那么整个网络的输出可以取任意值。

6）训练方法及其参数选择

针对不同的应用，BP 网络提供了多种训练、学习方法，以及如何选择训练函数和学习函数及其参数等。

2．BP 网络应用举例

1）用于模式识别与分类的 BP 网络

【例 4-9】以 BP 神经网络实现对图 4-29 所示的两类模式的分类。

根据图 4-29 所示的两类模式可以看出，分类为简单的非线性分类。有 1 个输入向量，包含 2 个输入元素；两类模式，1 个输出元素即可表示；可以以图 4-30 所示的两层 BP 网络来实现分类。

根据图 4-29 所示两类模式确定的训练样本为

$$\boldsymbol{p}=\begin{bmatrix}1 & -1 & -2 & -4\\ 2 & 1 & 1 & 0\end{bmatrix},\quad \boldsymbol{t}=[0.2\quad 0.8\quad 0.8\quad 0.2]$$

其中：因为 BP 网络的输出为 logsig 函数，所以目标向量的取值为 0.2 和 0.8，分别对应两类模式。在程序设计时，通过判决门限 0.5 区分两类模式。

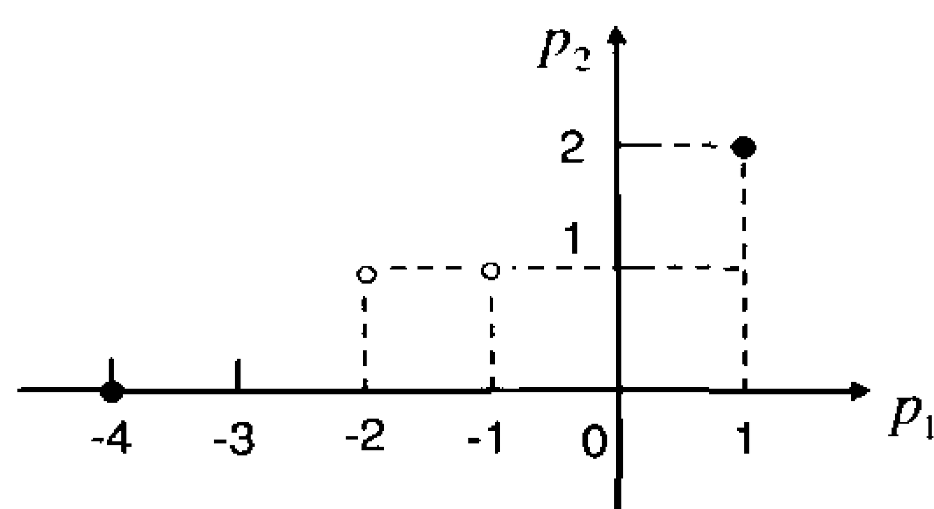


图 4-29　例 4-9 待分类模式

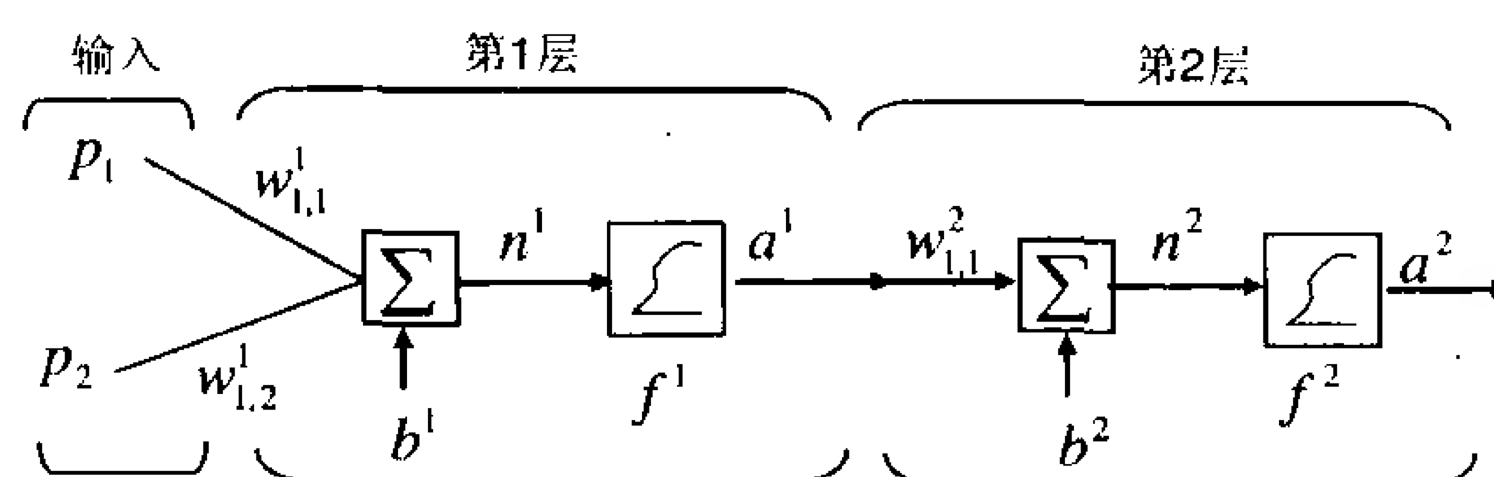


图 4-30　两层 BP 网络

因为处理的问题简单，所以采用最速下降 BP 算法（traingd 训练函数）训练该网络，以熟悉该算法的应用。

```
%创建和训练 BP 的 MATLAB 程序
clear all
%定义输入向量和目标向量
P=[1 2;-1 1;-2 1;-4 0]';
T=[0.2 0.8 0.8 0.2];
%创建 BP 网络和定义训练函数及参数
net=newff([-1 1;-1 1],[5 1],{'logsig' 'logsig'},'traingd');
net.trainParam.goal=0.001;
net.trainParam.epochs=5000;
%训练神经网络
[net,tr]=train(net,P,T);
%输出训练后的权值和阈值
iw1=net.iw{1};
b1=net.b{1}
iw2=net.lw{2}
b2=net.b{2}
```

BP 网络的初始化函数的默认值为 initnw，该初始化函数的详细说明见第 3 章。在本例中将随机初始化权值和阈值，所以每次运行上述程序的结果将不相同。当达不到要求时，可以反复运行以上程序，直到满足要求为止。其中的一种运行结果如下

```
TRAINGD, Epoch 0/5000, MSE 0.0242271/0.001, Gradient 0.0402598/1e-010
TRAINGD, Epoch 25/5000, MSE 0.0238301/0.001, Gradient 0.0394322/1e-010
TRAINGD, Epoch 50/5000, MSE 0.0234489/0.001, Gradient 0.0386496/1e-010
……
TRAINGD, Epoch 4950/5000, MSE 0.00359364/0.001, Gradient 0.0112599/1e-010
TRAINGD, Epoch 4975/5000, MSE 0.00356209/0.001, Gradient 0.0112065/1e-010
TRAINGD, Epoch 5000/5000, MSE 0.00353084/0.001, Gradient 0.0111531/1e-010
TRAINGD, Maximum epoch reached, performance goal was not met.
iw1 =
   -1.3176   -6.1050
    6.3150   -1.2373
    5.0119    3.7545
   -1.3784   -6.1587
    4.7074   -4.0190
b1 =
    6.2780
   -2.7471
   -0.0059
```

```
   -3.0678
    6.3720
iw2 =
   -3.1540    -3.1972     0.4336    -2.4313    -3.4589
b2 =
    4.1320
```

误差性能曲线如图 4-31 所示，从曲线上可以看出，训练经过了 5000 次仍未达到要求的目标误差 0.001，说明采用训练函数 traingd 进行训练的收敛速度是很慢的。

虽然训练的误差性能未达到要求的目标误差，但这并不妨碍我们以测试样本对网络进行仿真。

```
%网络仿真的 MATLAB 程序
p=[1 2;-1 1;-2 1;-4 0]';
a2=sim(net,p1)
a2=a2>0.5
a2 =
      0      1      1      0
p1 =
      1     -1     -2     -4
      2      1      1      0
```

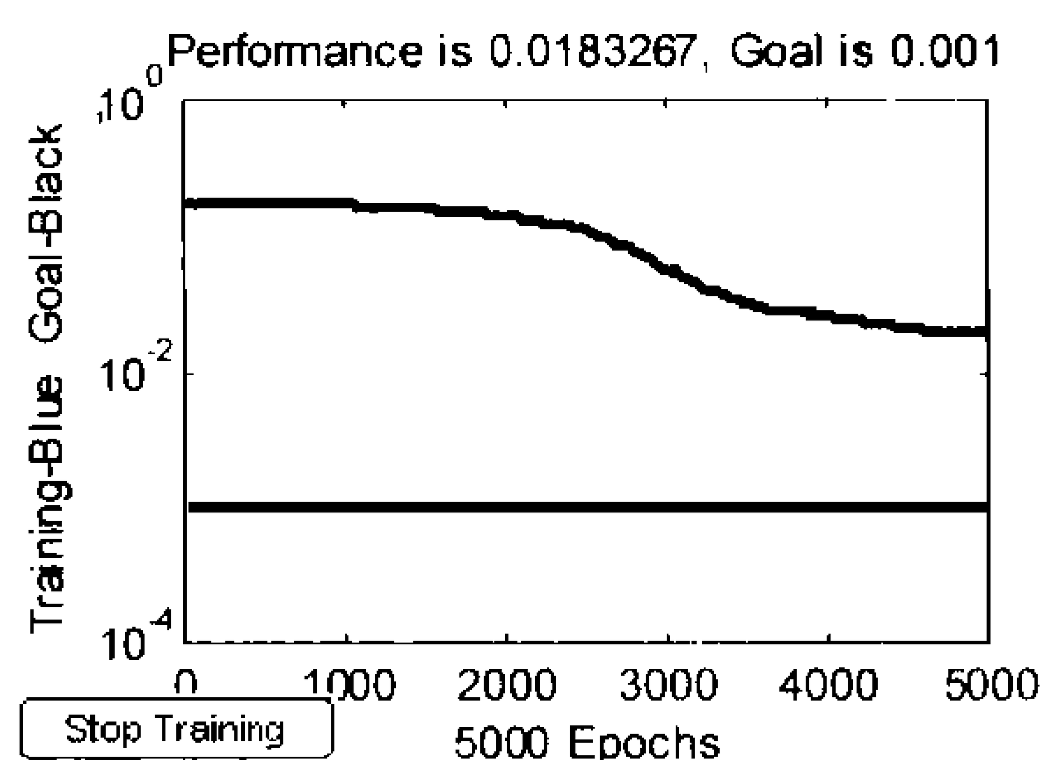


图 4-31　误差性能曲线

结果表明可以完成上述两类模式的分类。

2）去除噪声的 BP 网络

【例 4-10】利用 BP 神经网络去除噪声问题。

在 MATLAB 神经网络工具箱中，提供了 26 个大写字母的数据矩阵，利用 BP 神经网络，可以进行字符识别处理。

输入训练样本数据和测试样本。

```
%训练样本数据点
[AR,TS]=prprob;
A=size(AR,1);
B=size(AR,2);
C2=size(TS,1);
%测试样本数据点
CM=AR(:,13)
noisyCharM=AR(:,13)+rand(A,1)*0.3
figure
plotchar(noisyCharM)
```

BP 网络训练采样全训练样本集，即使用所有 26 个大写字母，测试样本采样包含噪声的字母 M 数据点。字母 M 和对应包含噪声的字母 M 图示如图 4-32 所示。

创建 BP 神经网络，并使用全训练数据点训练 BP 神经网络。

```
%创建 BP 网络，并使用数据点训练网络
P=AR;
T=TS;
%输入层包含 10 个神经元，输出层为 C2 个神经元，
    %输入、输出层都使用 logsig 传递函数
net=newff(minmax(P),[10,C2],{'logsig'
'logsig'},'traingdx');
net.trainParam.show=50;
net.trainParam.lr=0.1;
net.trainParam.lr_inc=1.05;
net.trainParam.epochs=3000;
net.trainParam.goal=0.01;
[net,tr]=train(net,P,T);
```

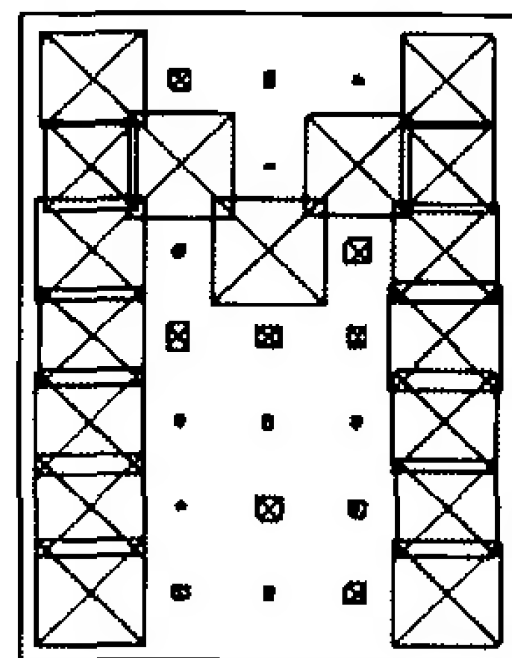


图 4-32　字母 M 和包含噪声的字母 M 图示

训练过程曲线如图 4-33 所示。

回代检验和测试样本点的检验如下。

```
%回代检验
A=sim(net,CM);
%测试样本检验
a=sim(net,noisyCharM);
%找到字母所在位置
pos=find(compet(a)==1)
figure
%绘制去除噪声后的字母
plotchar(AR(:,pos))
```

包含噪声的字母 M 经过 BP 网络后，输出结果如图 4-34 所示。BP 网络去除了字母 M 上的随机噪声。

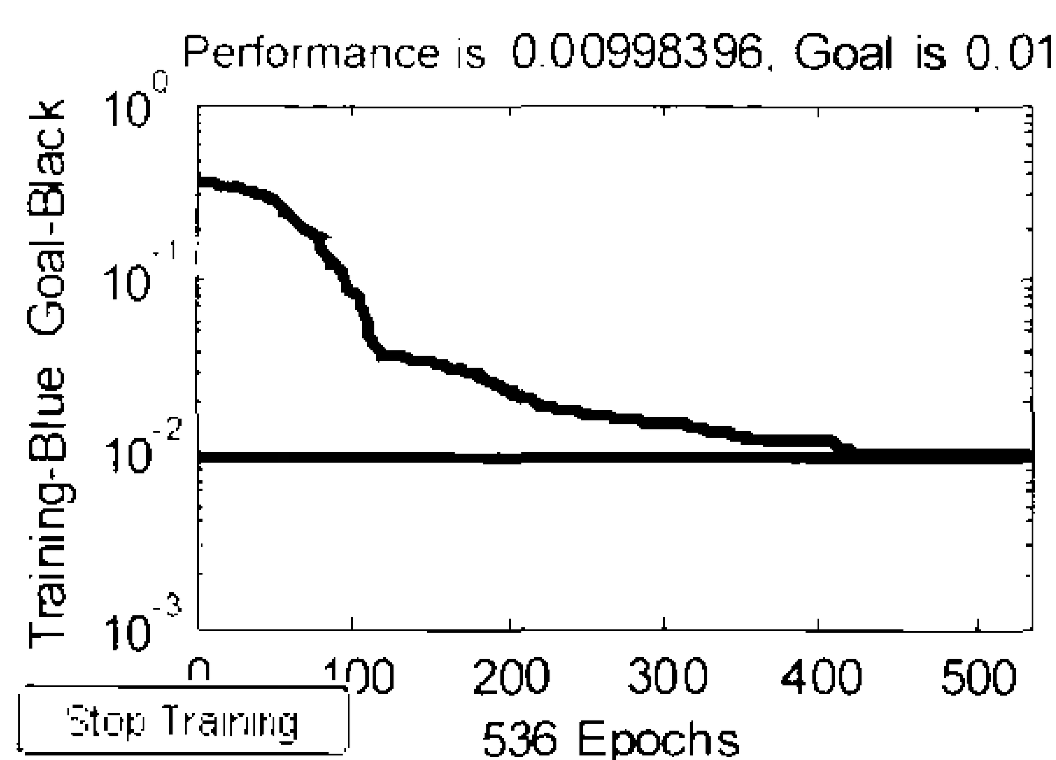


图 4-33　BP 网络训练过程曲线

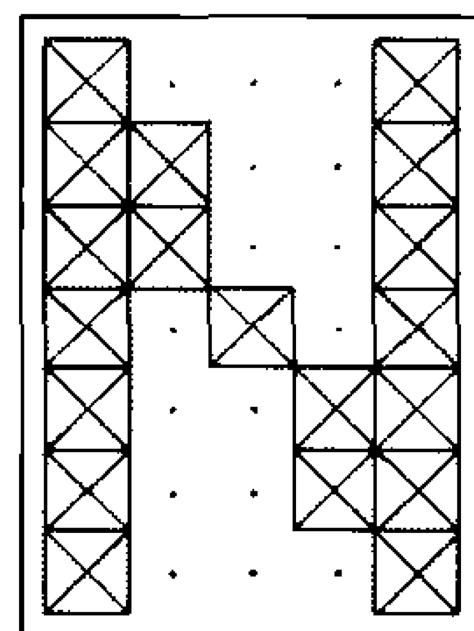
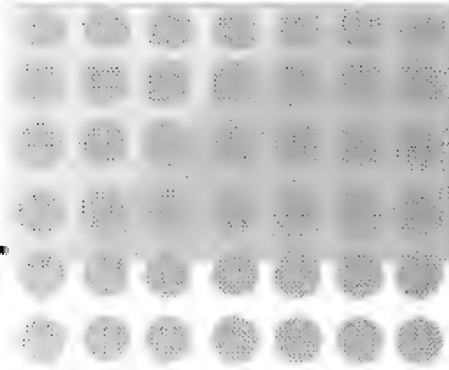


图 4-34　包含噪声的字母 M 经过 BP 网络训练后的输出结果

3）用于曲线拟合的 BP 网络

在实际应用中，往往希望产生一些非线性的输入/输出曲线，且没有明确的函数关系，借助神经网络实现曲线拟合，可以很方便地解决这一问题。

【例 4-11】已知某系统输出 y 与输入 x 的部分对应关系如表 4-3 所示。设计一 BP 神经网络，



完成 $y = f(x)$。

表 4-3　函数 $y = f(x)$ 的部分对应关系

x	-1	-0.9	-0.8	-0.7	-0.6	-0.5	-0.4	-0.3	-0.2	-0.1
y	-0.832	-0.423	-0.024	0.344	1.282	3.456	4.02	3.232	2.102	1.504
x	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
y	0.248	1.242	2.344	3.262	2.052	1.684	1.022	2.224	3.022	1.984

以隐层节点数为 15 的单输入和单输出两层 BP 网络来实现曲线拟合。

创建和训练 BP 网络的 MATLAB 程序。

```
clear all;
P=-1:0.1:0.9;
T=[-0.832 -0.423 00.024 0.344 1.282 3.456 4.02 3.232 2.102 1.504...
    0.248 1.242 2.344 3.262 2.052 1.684 1.022 2.224 3.022 1.984];
net=newff([-1 1],[15 1],{'tansig' 'purelin'},'traingdx','learngdm');
net.trainParam.epochs=2500;
net.trainParam.goal=0.001;
net.trainParam.show=10;
net.trainParam.lr=0.05;
net=train(net,P,T);
```

训练结果如下。

```
TRAINGDX, Epoch 0/2500, MSE 11.0032/0.001, Gradient 16.8035/1e-006
......
TRAINGDX, Epoch 284/2500, MSE 0.000983867/0.001, Gradient 0.00777523/1e-006
TRAINGDX, Performance goal met.
```

训练的误差性能曲线如图 4-35 所示。

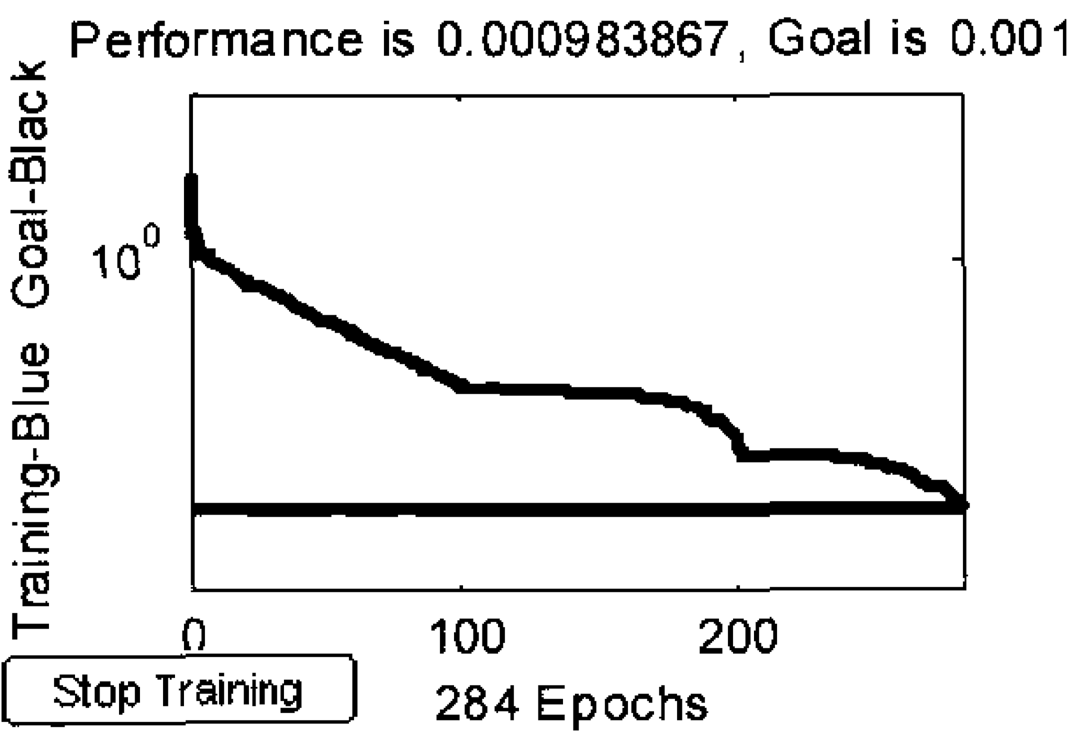
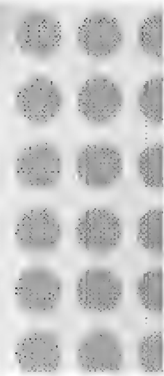


图 4-35　训练的误差性能曲线

BP 网络仿真的 MATLAB 程序如下。

```
P=-1:0.1:0.9;
T=[-0.832 -0.423 00.024 0.344 1.282 3.456 4.02 3.232 2.102 1.504...
    0.248 1.242 2.344 3.262 2.052 1.684 1.022 2.224 3.022 1.984];
hold on
plot(P,T,'r+');
p=-1:0.01:0.9;
```



```
R=sim(net,p);
plot(p,R);
hold off
```

曲线拟合如图 4-36 所示。实线为得到的拟合曲线；“+”为训练样本。从结果上看，可以对个别训练样本进行很好的拟合，但拟合曲线欠光滑，出现了“过适配”现象。如果改用 trainbr 训练函数进行训练，则曲线拟合会变得更光滑些，在此希望读者自行动手试一试。

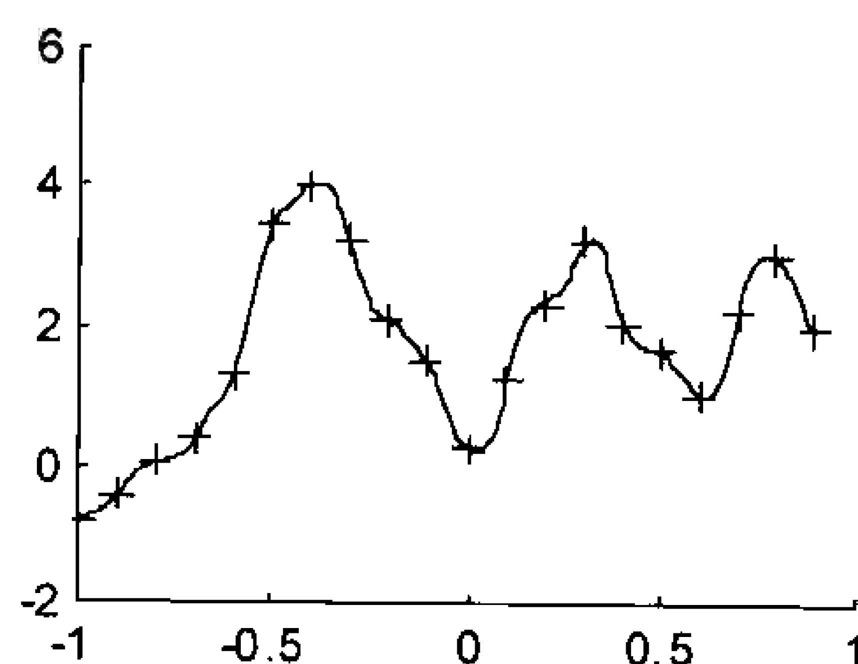


图 4-36　用 sim 函数的曲线拟合结果

4.4　径向基网络

众所周知，BP 网络用于函数逼近时，权值的调节采用的是负梯度下降法。这种调节权值的方法有其局限性，即收敛速度慢和局部极小等。本节主要介绍逼近能力、分类能力和学习速度等方面均优于 BP 网络的另一种网络——径向基函数网络（Radial Basis Function，RBF）。

4.4.1　径向基函数网络模型

径向基函数网络是一种两层前向型神经网络，包含一个具有径向基函数神经元的隐层和一个具有线性神经元的输出层。

1．径向基函数神经元模型

图 4-37 所示为一个有 R 个输入的径向基神经元模型。

径向基函数神经元的传递函数有各种各样的形式，但最常用的形式是高斯函数（radbas）。与前面介绍的神经元不同，神经元 radbas 的输入为输入向量 $\boldsymbol{P}$ 和权值向量 $\boldsymbol{w}$ 之间的距离乘以阈值 b 。径向基传递函数可表示为如下形式

$$\text{radbas}(n) = e^{-n^2} \tag{4-46}$$

径向基函数的图形如图 4-37 所示。

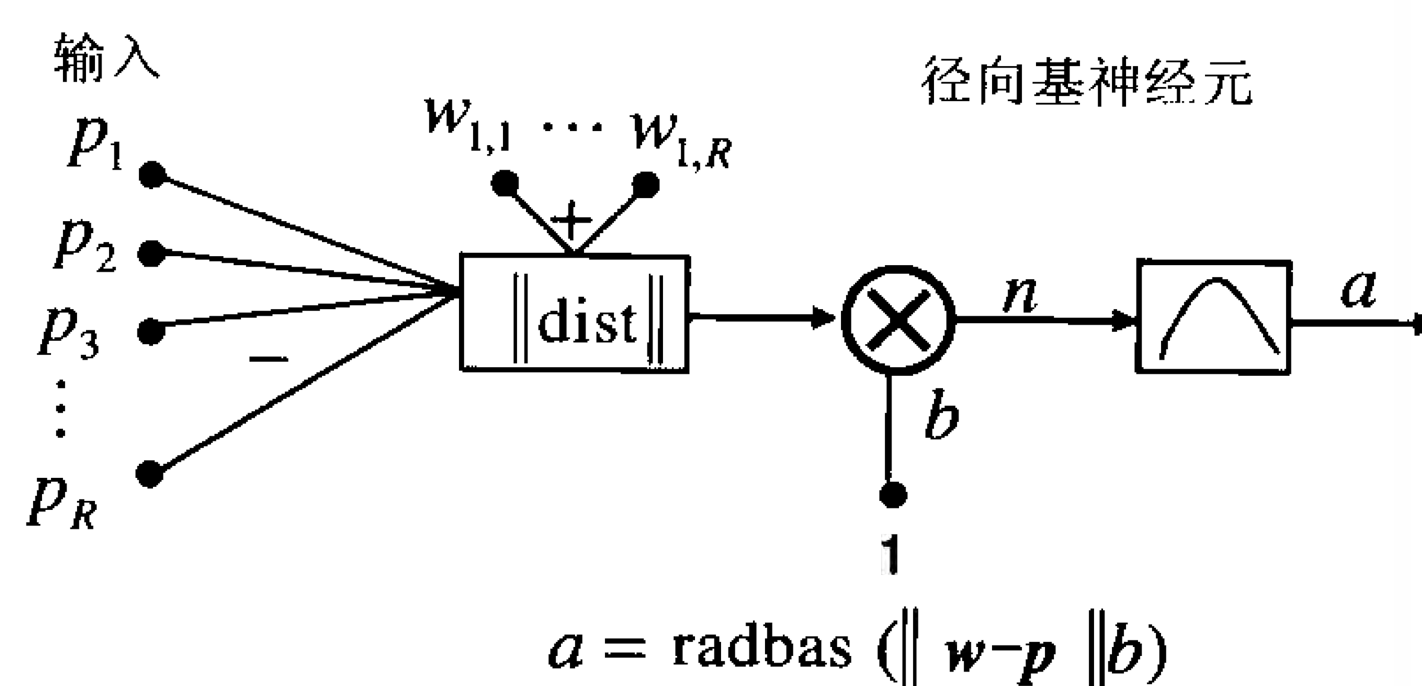


图 4-37　径向基神经元模型

径向基函数的图形如图 4-38 所示。

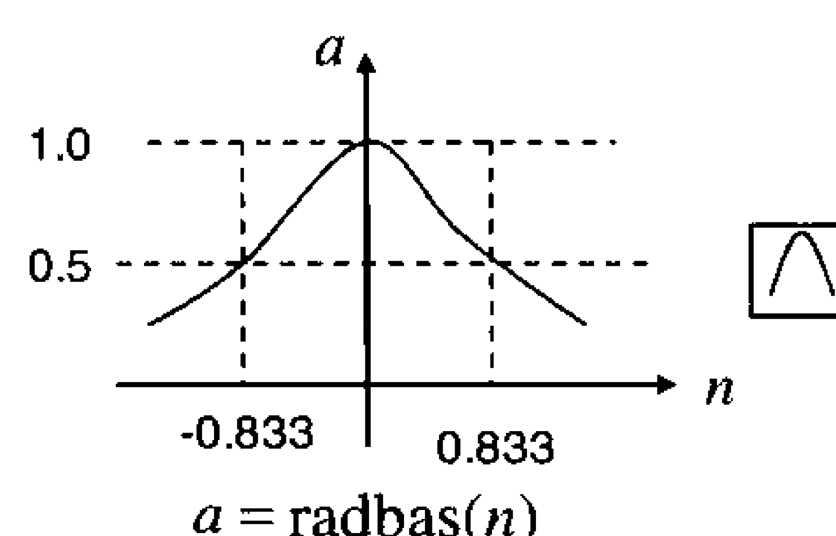


图 4-38　径向基函数

从图 4-38 中可以看出，n 为 0 时，径向基函数的输出最大值为 1，即权值向量 $\boldsymbol{w}$ 和输入向量 $\boldsymbol{p}$ 之间距离减小时，输出就会增加。也就是说，径向基函数对输入信号在局部产生响应。函数的输入信号 n 靠近函数的中央时，隐层节点将产生较大的输出。由此可以看出这种网络具有局部逼近能力，所以径向基函数网络也称为局部感知场网络。阈值 b 用于调整径向基神经元的敏感度，例如，假设神经元阈值 b =0.1，那么对于任意与权值矢量 $\boldsymbol{w}$ 之间距离为 8.33 的输入向量 $\boldsymbol{p}$，其输出都是 0.5。

2．径向基函数的网络结构

径向基函数网络包括隐层和输出层。输入信号传递到隐层，隐层有 S^1 个神经元，节点函数为高斯函数；输出层有 S^2 个神经元，节点函数通常是简单的线性函数。其结构如图 4-39 所示。

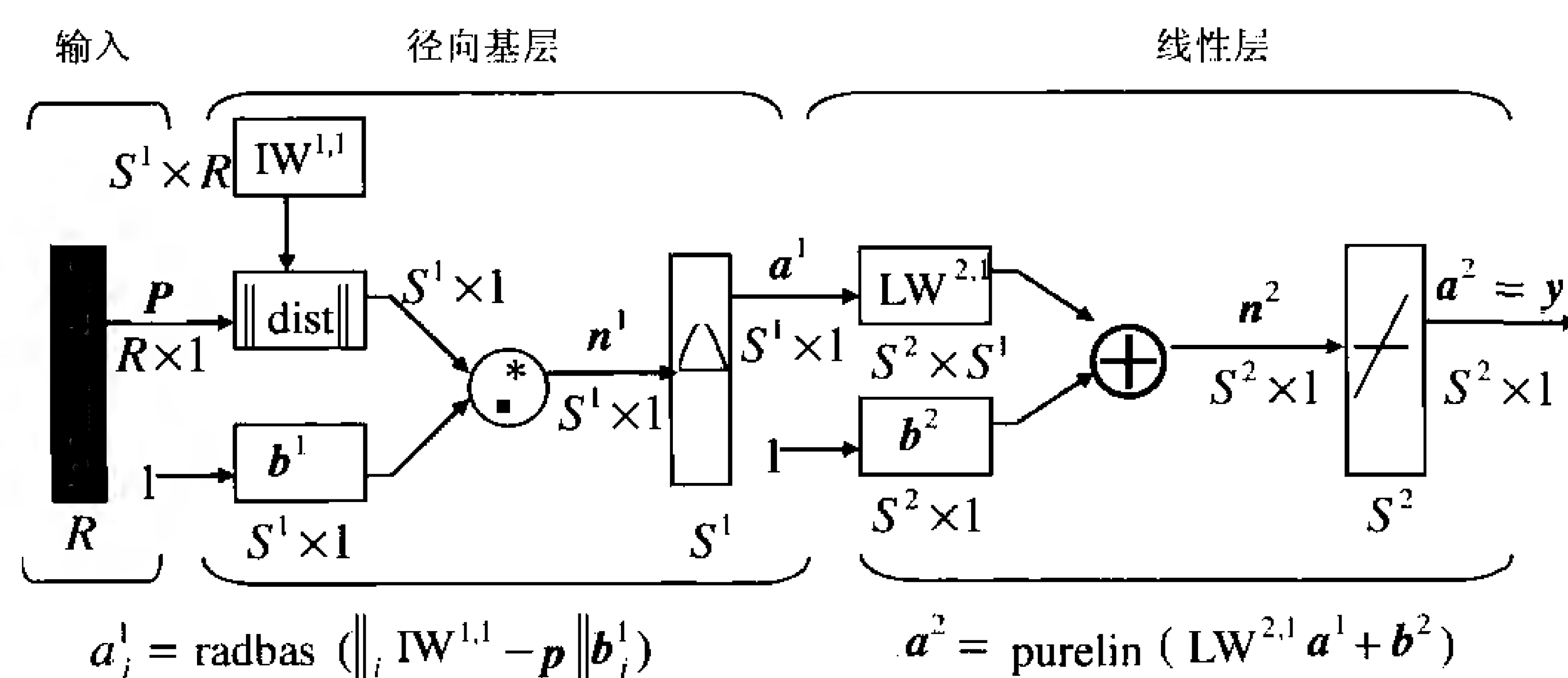


图 4-39　径向其函数网络的结构

其中，R 为输入向量元素的数目；S^1 为第一层神经元的数目；S^2 为第二层神经元的数目；a_j^1 为向量 $\boldsymbol{a}^1$ 的第 j 个元素；${}_i\mathrm{IW}^{1,1}$ 为权值矩阵 $\mathrm{IW}^{1,1}$ 的第 i 个向量。

‖dist‖ 模块计算输入向量 $\boldsymbol{P}$ 和输入权值 $\mathrm{IW}^{1,1}$ 的行向量之间的距离，产生 S^1 维向量，然后与阈值 $\boldsymbol{b}_1$ 相乘，再经过径向基传递函数从而得到第一层输出。第一层输出 $\boldsymbol{a}^1$ 可由下面语句得到：

```
a{1}=radbas (netprod (dist (net.IW{1,1},p),net.b{1}))
```

实际上，函数 newrbe 和 newrb 在设计过程中已经包含了这些细节，所以可以直接应用，然后再通过 sim 函数得到网络的输出。

3．径向基函数网络的工作原理

当输入向量加到网络输入端时，径向基层的每个神经元都会输出一个值，代表输入向量与神经元权值向量之间的接近程度。

（1）如果输入向量与权值向量相差很多，则径向基层的输出接近于 0，经过第二层的线性神经元，输出也接近于 0。

（2）如果输入向量与权值向量很接近，则径向基层的输出接近于 1，经过第二层的线性神经元，输出值就更靠近第二层权值。

在这个过程中，如果只有一个径向基神经元的输出为 1，而其他的神经元输出均为 0 或者接近于 0，那么线性神经元层的输出就相当于输出为 1 的神经元相对应的第二层权值的值。一般情况下，不止一个径向基神经元的输出为 1，所以输出值也就会有所不同。

下面解析一下第一层的工作过程。

如图 4-39 所示，第一层神经元的网络输入为加权输入与相应的阈值的乘积，然后通过神经元函数 radbas 计算得到第一层神经元的网络输出。其中，加权输入表示输入向量与权值向量相等，加权输入即为 0，则第一层网络输入也为 0，那么第一层输出必然是 1；如果神经元的权值向量与输入向量之间的距离恰为散布常数spread值，则加权输入为spread，网络输入则为0.8325，那么第一层神经元输出为 0.5。

4.4.2 径向基函数网络的构建

1．径向基函数网络的严格设计

应用 newrbe 函数可以快速设计一个径向基函数网络，且使得设计误差为 0，调用方式可参考第 3 章 newrbe 的解析。

由 newrbe 函数构建的径向基函数网络，径向基层（第一层）神经元数目等于输入向量的个数。径向基层的阈值设定为 0.8326/spread，目的是使得加权输入为 ± spread 时径向基层输出为 0.5，阈值的设置决定了每个径向基神经元对于输入向量产生响应的区域。例如，如果 spread 为 4，每个径向基神经元对应那些不小于 0.5 的第一层输出，其输入向量均在距离权值向量为 4 的区域内。由此看来，spread 应当足够大，使得神经元响应区域覆盖所有输入区域。

径向基网络的第二层为线性神经元层，其权值和阈值的调整需要满足输出向量与期望值相等的要求。表达语句为

[W{2,1} b{2}]*[A{1};ones]=T

其中，A{1}为第一层输出；W{2,1}为第二层权值，b{2}为第二层阈值，T 为期望值。如果计算第二层的权值和阈值，需要使得均方误差最小，计算调用方式如下

Wb = T/[P;ones(1,Q)]

其中，Wb 包含了权值和阈值的向量。

这个问题中，如果输入/输出样本有 R 个，问题有 R 个约束，而变量个数为 $R+1$ 个（R 个神经元的 R 个权值和一个阈值）。对于变量数目多于约束条件数目的线性问题，其解为无穷多个。

应用 newrbe 函数构建一个零误差网络，其散布常数 spread 的设置是一个关键问题。如前所述，spread 需要足够大才能覆盖所有的输入区间，但是如果 spread 太大，则每个神经元的响应区域又会交叉过多，反而带来精度问题。

2．更有效的径向基函数网络的设计

应用 newrbe 函数设计网络时，径向基神经元的数目与输入向量的个数相等，那么在输入向量较多的情况下，则需要很多的神经元，这就给网络设计带来一定的难度。函数 newrb 则能更有效地进行网络设计。

用径向基函数网络逼近函数时，newrb 函数可自动增加网络的隐层神经元数目，直到均方误差满足精度要求或者神经元数目达到最大为止。其调用方式参数第 3 章。

函数 newrb 的设计方法与函数 newrbe 类似，唯一不同的是 newrb 函数每一次循环只产生一个神经元，而每增加一个径向基神经元，都能最大程度地降低误差，如果未达到精度要求则继续增加神经元，满足精度要求后则网络设计成功。程序终止条件是满足精度要求或者达到最大神经元数目。

应用 newrb 函数进行径向基函数神经网络设计时，散布常数是一个非常重要的参数，后面的实例将演示散布常数对网络设计的影响。

径向基网络和普通的前向网络有所不同，隐层神经元是径向基神经元而不是 tansig 或者 logsig 神经元，应用径向基网络设计有其特有的优势。

（1）普通的前向网络中 sigmoid 神经元能够覆盖较大的输入区域，但是普通前向网络神经元数目在训练前就已经固定下来。而径向基神经元虽然只对相对较小的区域产生响应，但是在输入区间较大时，可以适当地增加径向基神经元来调整网络，从而达到精度要求。

（2）径向基函数网络的设计比普通前向网络训练要省时得多。

4.4.3 RBF 网络应用实例

这里将 RBF 网络结构应用于某地的地下水动态模拟与预测，演示训练样本集与检测样本集的构建、原始数据的预处理、神经网络的构建训练和检测及结构评价的整个过程。

1．前期准备

地下水位主要受河道流量、气温、饱和差、降水量和蒸发量等重要因素的影响。由此从资料中归纳出 24 组数据，如表 4-4 所示。选定其中的 1～19 组作为训练样本，20～24 组作为测试样本。

表 4-4　地下水位及其影响因子监测数据表

序　号	河道流量	气　温	饱和差	降水量	蒸发量	水　位
1	0.0177	0	0.0200	0.0054	0.0580	0.6725
2	0.0230	0	0.1000	0.0054	0	0.6943
3	0.0619	0.2424	0.1500	0.0323	0.2319	0.6376
4	0.2212	0.6061	0.4000	0.1613	0.5217	0.4891
5	0.0796	0.8182	0.8000	0.0968	0.7971	0.1616
6	0.1504	0.9697	0.9000	0.6075	0.8406	0.0699
7	0.1681	1.0000	0.7000	0.1559	0.6957	0.1092
8	0.1504	0.9394	0.5000	0.3978	0.5507	0.1048
9	0.1150	0.7576	0.4000	0.1129	0.2174	0.2533
10	0.1593	0.5606	0.4000	0.0806	0.3913	0.4017
11	0.0885	0.3030	0.5200	0.0753	0.2193	0.6201
12	0.2035	0.3182	0.3500	0.0591	0	0.6638
13	0	0.3333	0.1000	0.0054	0.0290	0.5764
14	1.0000	0.9697	0.4500	1.0000	0.6812	0.0437
15	0.6106	0.8788	0.4000	0.6129	0.5507	0
16	0.6814	0.6970	0.4000	0.3226	0.4058	0.0568
17	0.3982	0.4848	0.2000	0.1882	0.2609	0.2009
18	0.1858	0.3333	0.1000	0.0215	0.1304	0.4105
19	0.0708	0.0909	0	0.0323	0.0290	0.5153
20	0.0442	0.0909	0.1500	0.0108	0.0725	0.6070
21	0.1150	0.3030	0.2000	0.0215	0.4783	1.0000
22	0.1681	0.6061	0.6000	0	0.3478	0.4148
23	0.0708	0.8485	0.9000	0.1022	0.8261	0.1921
24	0.1327	0.9545	1.0000	0.4355	1.0000	0.0873

注意

获得有关地下水位的数据后，在用于训练样本和测试样本之前，需要进行归一化处理。表 4-4 中的数据为已经归一化处理的数据。

2. 网络的创建、训练和测试

RBF 网络的输入层神经元个数取决于地下水位影响因子的个数，由表 4-4 可知，其个数为 5。由于输出就是地下水位的值，所以输出层神经元个数为 1。利用函数 newrbe 创建一个精确的神经网络，该函数在创建 RBF 网络时，自动选择隐含层的数目，使得误差为 0。MATLAB 代码为

```
SPREAD=1.5;
net=newrbe(P,T,SPREAD);
```

其中，***P*** 为输入向量，***T*** 为目标向量，它们可从表 4-4 中得到。SPREAD 为径向基函数的分布密度，SPREAD 越大，函数越平滑，这里先取 1.5。由于网络的建立过程就是训练过程，因此，此时得到的网络 net 已经是训练好的了。

接下来对网络进行仿真，验证其预测性能。代码为

```
y=sim(net,P_test)
```

其中，P_test 为网络的测试样本，可从表 4-4 中得出。运行结果为

```
y =
     0.6457     1.0825     0.3778     0.0052     0.1849
```

经过反归一化处理，可得到水位

H=6.8581 7.8632 6.2538 5.3947 5.8007

同实际值 H0=6.77 7.67 6.33 5.82 5.58 相比较，可得出预测误差，对于地下水位的预报来说，网络的预报误差并不大。

此外，SPREAD 值的大小影响网络的预测精度。接下来，分别在 SPREAD=2、3、4 或 5 的情况下计算网络的预报精度。代码为

```
y=rand(4,5);
for i=1:4
    net=newrbe(P,T,i+1);
    y(i,:)=sim(net,P_test);
end
plot(1:5,y(1,:)-t_test,'b');
hold on
plot(1:5,y(2,:)-t_test,'r--');
hold on
plot(1:5,y(3,:)-t_test,'g*');
hold on
plot(1:5,y(4,:)-t_test,'.');
hold on
y_bp=[0.1201 -0.2559 -0.1828 -0.1237 0.0001];
plot(1:5,y_bp,'*');
hold off
```

结果如图 4-40 所示。可以看出，当 SPREAD=2 或 3 时，网络的预报误差最小。因此，本例中 SPREAD=2 或 3 都可得到理想的结果。图中“*”号表示为采用 BP 网络进行地下水位预报的误差。

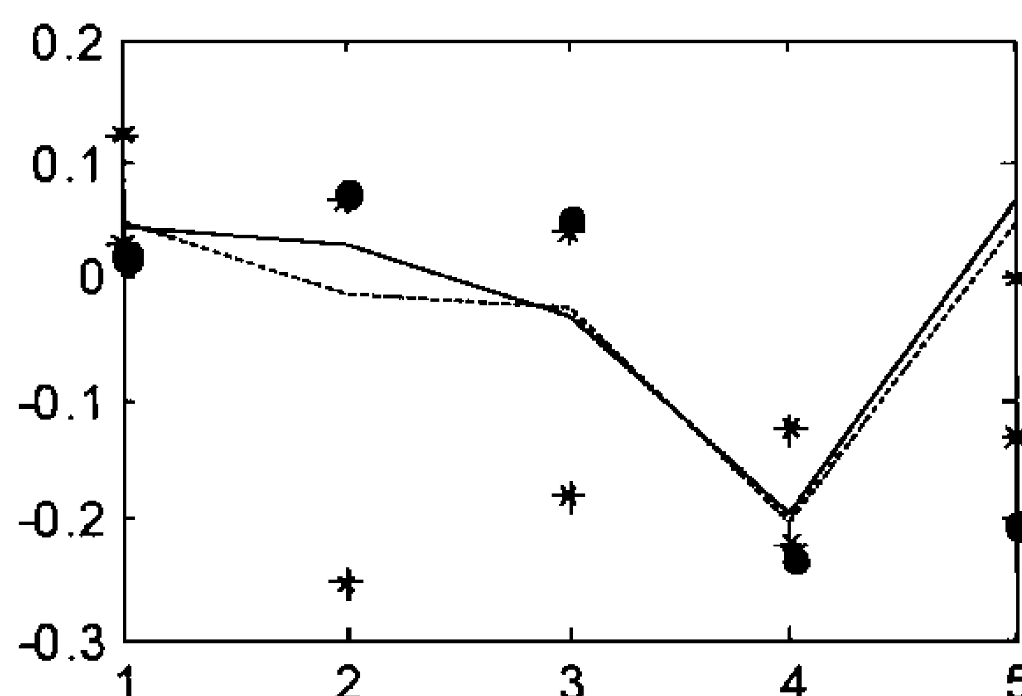


图 4-40　SPREAD 取不同值的预报误差

为了验证 RBF 网络相对于 BP 网络的优势，在本例中利用 BP 网络对地下水位进行重新预报。选择的 BP 网络为 5×11×1 的结构，训练函数为 trainlm。经过 500 次训练后，对网络进行仿真，并计算网络的预测误差。误差变量用 y_bp 表示。综合对比后发现，对于预报精度来说，BP 网络明显不如 RBF 网络，而且 BP 网络的训练时间明显大于 RBF 网络，其训练速度比较慢。

4.4.4　RBF 网络的非线性滤波

1．非线性滤波

早期的数字信号处理和数字图像处理主要以线性滤波器为主要处理手段。线性滤波器由于数学表达式比较简单并且具有其他一些比较理想的特性，所以，实现起来相对比较容易。然而，当信号中存在由系统非线性引起的噪声或非高斯叠加型噪声时，线性滤波器便不能很好地工作。目前，最优非线性滤波存在“实时”问题，即一、滤波器权系数的实时计算；二、非线性滤波器的实时实现。描述系统的非线性差分方程为

状态方程：$\boldsymbol{x}(n+1)=f(\boldsymbol{x}(n))+\boldsymbol{v}(n)$

观测方程：$\boldsymbol{y}(n+1)=h(\boldsymbol{x}(n))+\boldsymbol{w}(n)$

其中，f 和 h 都是非线性函数，$\boldsymbol{w}(n)$ 和 $\boldsymbol{v}(n)$ 为零均值的白噪声序列。

所谓最优滤波，就是解决从观测值 $\boldsymbol{y}(n)$ 估计出状态 $\hat{\boldsymbol{x}}(n)$，且使得 $\hat{\boldsymbol{x}}(n)$ 可以最好地接近 $\boldsymbol{x}(n)$ 的问题。RBF 网络具有唯一的最佳逼近特性，因此尝试将其应用于最优滤波，即利用已知的采样数据对非线性函数作最优逼近。由 RBF 网络的输入/输出表达式可得 h 的估计值

$$\hat{h}=\sum_{i=1}^{N} w_i R_i(\cdot)=\boldsymbol{W}^{\mathrm{T}}\boldsymbol{r}(\cdot) \tag{4-47}$$

其中，$\boldsymbol{W}=[w_i]_{i=1}^{N}$，$\boldsymbol{r}=[R_i]_{i=1}^{N}$，$N$ 为训练次数。接下来，设计一个 RBF 网络，使得它可以在规定的精度内逼近 h。

2．网络设计

输入样本为 $\boldsymbol{P}$，目标向量为 $\boldsymbol{T}$，如下所示。

```
P=[-1:0.1:1];
T=[-0.9602 0.5770 0.0729 0.3771 0.6405 0.6600 0.4609 0.1336 -0.2013 -0.4344 -0.5000
    -0.3930 0.1647 0.0988 0.3072 0.3960 0.3449 0.1816 -0.0312 -0.2189 -0.3201];
for i=1:5
```

```
    net=newrbe(P,T,i);
    y(i,:)=sim(net,P);
end
```

在上面的代码中，我们利用 RBF 网络的精确创建函数 newrbe，创建了一个准确的 RBF 网络，它已经可以逼近目标向量了。由于径向基函数的分布密度 SPREAD 可以影响网络的精度，因此，这里将其设定为 1、2、3、4 和 5，共 5 个整数，观察它们对网络预测性能的影响。

网络的逼近误差曲线如图 4-41 所示。由图可见，分布密度为 1 和 2 时，网络的逼近误差比较小，考虑到收敛速度和计算方面的原因，这里的分布密度选 1。

此时网络的输出结果为

```
y(1,:) =
-0.9587   0.5671   0.1014   0.3379   0.6597   0.6715 0.4480 0.1319 -0.1972   -0.4463   -0.4964   - 0.3111
-0.0089   0.2262   0.3166   0.3364   0.3448   0.2322 -0.0725    -0.2044      -0.3221
```

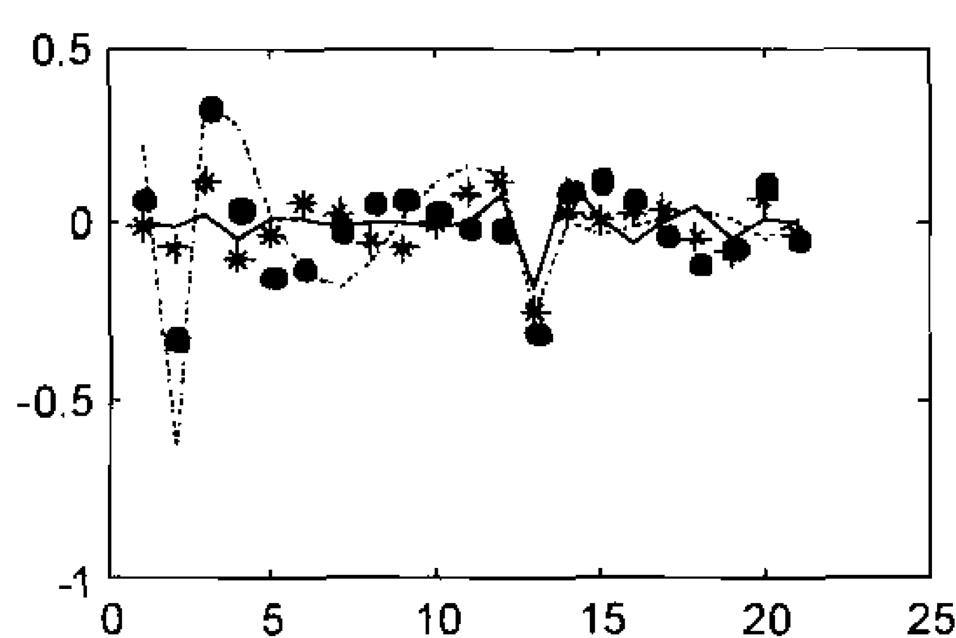


图 4-41　网络逼近误差曲线

本实例的完整 MATLAB 代码如下。

```
P=[-1:0.1:1];
T=[-0.9602 0.5770 0.0729 0.3771 0.6405 0.6600 0.4609 0.1336 -0.2013 -0.4344 -0.5000
   -0.3930 0.1647 0.0988 0.3072 0.3960 0.3449 0.1816 -0.0312 -0.2189 -0.3201];
%创建 5 个 RBF 网络，分布密度分别为 1、2、3、4、5
for i=1:5
    net=newrbe(P,T,i);
    y(i,:)=sim(net,P);
end
%绘制误差曲线
plot(1:21,y(1,:)-T);
hold on;
plot(1:21,y(2,:)-T,'*');
hold on;
plot(1:21,y(3,:)-T,'b.');
hold on;
plot(1:21,y(4,:)-T,'r--');
hold on;
plot(1:21,y(5,:)-T,'g-.');
hold off;
```

4.5　GMDH 网络

GMDH 的全称是 Group Method of Data Handling（数据处理的群方法）。GMDH 网络也称为

多项式网络，它是前馈神经网络中常用的一种用于预测的神经网络。它的特点是网络结构不固定，而且在训练过程中不断改变。

4.5.1　GMDH 网络理论

CMDH 网络的结构在训练过程中是变化的。如图 4-42 所示的是训练后的一个比较典型的 GMDH 网络。

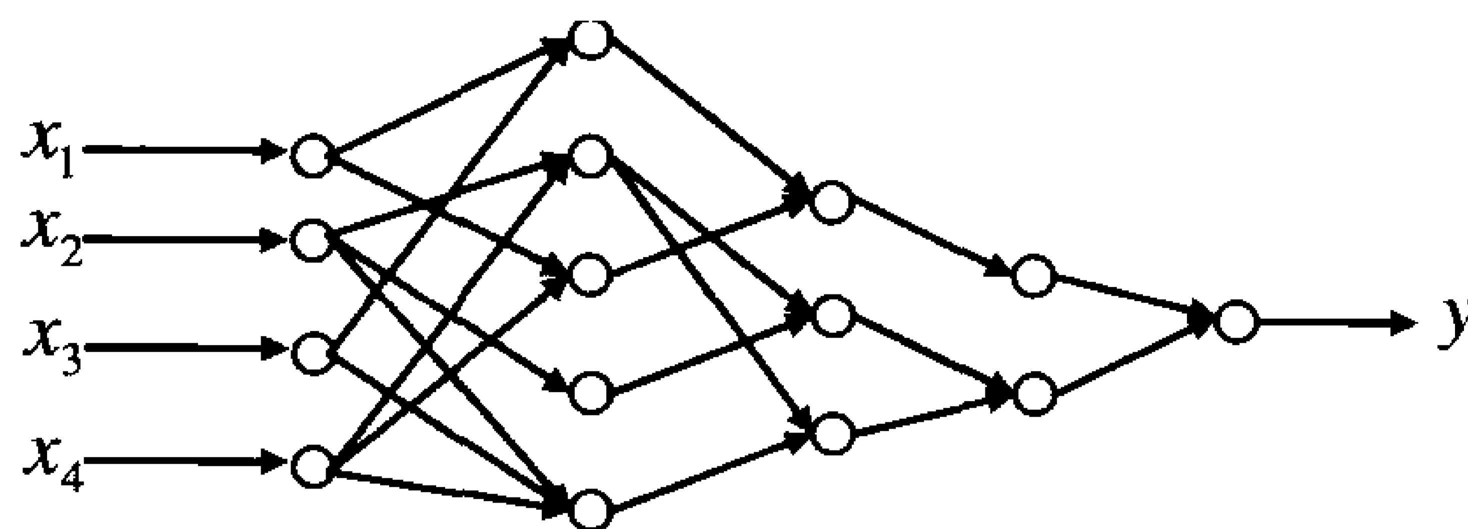


图 4-42　GMDH 网络结构

该网络有 4 个输入和 1 个输出。GMDH 网络的输入层将输入信号前向传递到中间层，中间层的每个神经元和前一层的 2 个神经元对应，因此，输出层的前一层（中间层）肯定只有 2 个神经元。

一般采用自适应线性元件作为 GMDH 网络中的神经元，如图 4-43 所示。该神经元的输入/输出关系为

$$Z_{k,l}=w_5Z_{k-1,i}^2+w_4Z_{k-1,i}Z_{k-1,j}+w_3Z_{k-1,j}^2+w_2Z_{k-1,i}+w_1Z_{k-1,j}+w_0 \qquad (4\text{-}48)$$

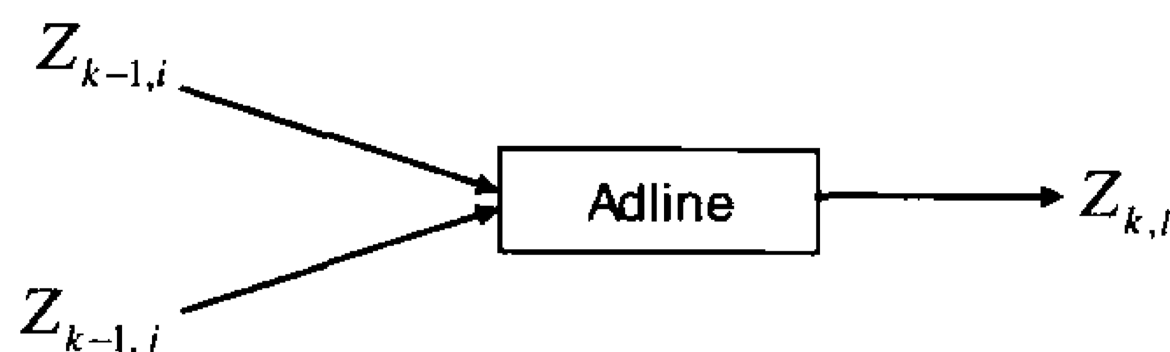


图 4-43　GMDH 网络中的神经元(输入层神经元除外)

其中，$Z_{k,l}$ 表示第 k 层的第 l 个处理单元，且 $z_{0,l}=x_l$，$w_i(i=1,2,3,4,5)$ 为神经元的权值。由上式可见，GMDH 网络中的处理单元的输出是 2 个输入量的二次多项式，因此网络的每一层将使得多项式的次数增大 2 阶，其结果是网络的输出可以表示成输入的高阶（$2k$ 阶）多项式，其中 k 是网络的层数（不含输入层）。

4.5.2　GMDH 网络的训练

训练一个 GMDH 网络，包括从输入层开始构造网络，调整每一个神经元的权值和增加网络层数直到满足映射精度为止。第一层的神经元数取决于输入信号的数量，每一个输入信号需要一个神经元。一般地，假定网络仅有一个输入，所以输入层只有一个神经元。假设在时刻 k 神经元的权值向量为

$$\boldsymbol{w}_k=[w_0,w_1,w_2,w_3,w_4,w_5]^{\mathrm{T}}$$

输入向量为

$$\boldsymbol{x}_k=[1,x_1,x_1^2,x_1x_2,x_2,x_2^2]$$

由 Widrow-Hoff 学习规则可知

$$\boldsymbol{w}_{k+1}^{\mathrm{T}}=\boldsymbol{w}_{k}^{\mathrm{T}}+\alpha\frac{\boldsymbol{x}_{k}}{\left|\boldsymbol{x}_{k}\right|^{2}}\left(\boldsymbol{y}_{dk}-\boldsymbol{w}_{k}\boldsymbol{x}_{k}^{\mathrm{T}}\right) \tag{4-49}$$

其中，$\boldsymbol{y}_{dk}$ 为神经元在 k 时刻的目标输出向量，α 为学习速率，取值在[0.1,1]之间。按照上式就可以调整神经元权值，降低神经元实际输出和目标输出之间的误差。

以上计算是在假定只有输入的前提下进行的。从权值调整公式可以看出，网络期望输出值 $\boldsymbol{y}_{dk}$ 出现在每个输入层神经元中，并希望通过训练使各神经元都能达到这一期望输出。对一个神经元来说，当训练数据集中每一个数据产生的均方差之和 S_{E} 达到最小时，对这个神经元的训练就结束，其权值予以固定。当输入层的神经元被全部训练一遍后，训练停止。这时，另一组数据（通常称为选择数据）被加到神经元上，并计算相应的 S_{E}。对那些 S_{E} 小于阈值的神经元，即放入下一层，而其余神经元则被舍弃，同时记录每一层神经元训练过程中产生的最小 S_{E}。若当前层在训练过程中产生的最小 S_{E} 小于前一层时（它表示网络精度得到提高），就产生一个新的神经元层，这一层中的神经元数取决于上一层中保留的神经元数，然后对新的神经元层进行训练和选择，而保持已训练的神经元层不变，这一过程一直进行到 S_{E} 不再减小为止。这时，取前一层神经元中误差最小的神经元的输出作为网络输出。当新的神经元层只有 1 个神经元，且该层的 S_{E} 小于前一层时，这一神经元就作为输出神经元。输出神经元确定以后，要对网络进行整理，所有与输出神经元无直接或间接联系的神经元都被舍弃，仅留下那些与输出有关的神经元。图 4-42 即是一个训练结束后的 GMDH 网络。

4.5.3 基于 GMDH 网络的预测

一般来说，所有的神经网络均可用于预测。但是，对于一般的前馈网络来说，在网络结构建立以后，其结构（神经元层数及每层神经元个数）都是固定的，在训练过程中不会有神经元的增加或减少。因此一个网络的性能好坏与事前确定的该网络结构是否合适有很大关系。

与此不同的是，GMDH 网络的结构是在训练中动态确定的。在训练过程中，网络的神经元层数不断增加，每增加一层就增加一些新的神经元，而那些性能不好的神经元则被舍弃，因而每一层中的神经元数也是可变的。

下面是训练一个用于预测的 GMDH 网络的步骤。

（1）数据预处理。包括数据规范化和除去数据中的静止直流成分。习惯上，对于已有的输入/输出样本，在进行神经网络的训练以前，首先进行归一化处理。

（2）决定网络的输入信号数。对于预测，需要用到 n 个过去输出值。如果需要，n 的值可以通过计算相关系数确定。

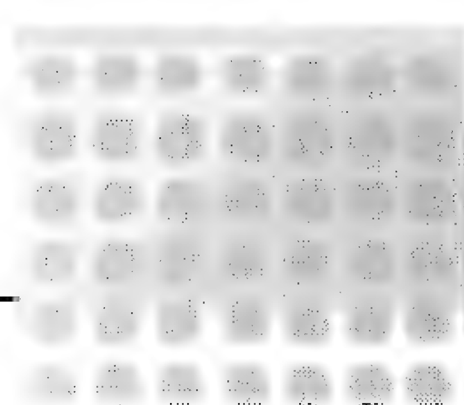
（3）将实验数据分成训练样本和预测样本。

（4）建立输入神经元层。神经元数与输入信号数 i 有关。对每个输入信号，都有一个神经元与之对应。因此，相应的神经元数为 C_i^2。

（5）将神经元权值的初始值设为 0。

（6）将训练数据组作用于输入层的每一个神经元。在 k 时刻取 $\boldsymbol{y}_{k-1}(k=1,2,\cdots)$ 作为输入信号，$\boldsymbol{y}_k$ 为期望输出，计算每一神经元的输入误差，并修正其权值和均方误差和，当均方差和大于上一循环计算值时，训练停止。

（7）输入选择数据，计算每一神经元的输出均方差。根据差值确定一个阈值，选择方差小

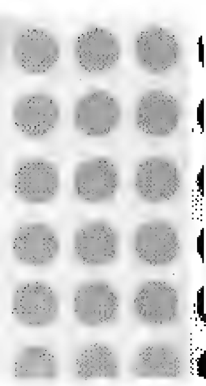


于阈值的神经元作为下一层神经元。

（8）当本层最小均方差大于前一层神经元的最小均方差时或本层仅有一个神经元时，停止训练过程。如果训练是由于最小方差偏大而停止的，则将前一层神经元作为输出层，并重新整理网络；若训练是因本层仅有一个神经元而停止的，且本次方差小于前一层时，则以本层神经元作为输出层并重新整理网络。所谓重新整理是指舍弃那些与输出神经元没有联系的神经元。

（9）利用评价数据组检查训练好的网络性能。评价数据组可以是上述样本数据和预测数据的结合，也可以是一组全新的数据。采用全新数据组可以在更广泛的基础上检查网络性能。

神经网络工具箱没有为 GMDH 网络提供有关的函数工具，因此，只有通过 MATLAB 的数学计算功能来实现以上算法。



第 5 章　自组织神经网络及 MATLAB 程序

前面的章节讲述的都是在训练过程中采用有教师学习方式的神经网络模型。这种学习方式在训练过程中，需要给网络提供期望输出，以便根据期望输出来调整网络的权重，使得实际输出和期望输出尽可能地接近。而本章要讲的自组织神经网络是一类采用无教师学习方式的神经网络模型。它无须期望输出，只是根据数据样本进行学习，并调整自身的权重以达到训练的目的，这也是自组织名称的由来。自组织神经网络的学习规则大都采用竞争型的学习规则，关于竞争型学习规则的理论，将在 5.1 节会有详细的介绍及举例说明。除了学习规则是以竞争形规则为主以外，自组织神经网络的结构也是不同的，有一维输出层的、二维输出层的、带有层反馈的等。模型不同，相应的竞争型学习算法也有变化，本章会一一介绍。

5.1　自组织竞争型神经网络

竞争型神经网络是基于无教师学习方法的神经网络的一种重要类型，它经常作为基本的网络形式，构成其他一些具有自组织能力的网络，如自组织映射网络、自适应共振理论网络、学习向量量化网络等。

生物神经网络存在一种侧抑制现象，即一个神经细胞兴奋后，通过它的分支会对周围其他神经细胞产生抑制，这种抑制使神经细胞之间出现竞争：在开始阶段，各神经元对相同的输出具有相同的响应机会，但产生的兴奋程度不同，其中兴奋最强的一个神经细胞对周围神经细胞的抑制作用也最强，从而使其他神经元的兴奋程度得到最大强度的抑制，而兴奋程度最强的神经细胞却“战胜”了其他神经元的抑制作用而脱颖而出，成为竞争的胜利者，并因为获胜其兴奋的程度得到进一步加强，正所谓“胜者为王，败者为寇”。竞争型神经网络在学习算法上，模拟了生物神经网络中神经元之间的兴奋、抑制与竞争的机制，进行网络的学习与训练。

5.1.1　竞争型神经网络模型

竞争型神经网络模型如图 5-1 所示。

可以看出竞争型神经网络为单层网络。$\|\text{ndist}\|$ 的输入为输入向量 $\boldsymbol{p}$ 和输入权值向量 IW，其输出为 $S^1\times 1$ 的列向量，列向量中的每个元素为输入向量 $\boldsymbol{p}$ 和输入权值向量 IW 距离的负数（negative），在神经网络工具箱中主要以距离函数 negdist 进行计算。

$\boldsymbol{n}^1$ 为竞争层传输函数的输入，其值为输入向量 $\boldsymbol{p}$ 和输入权值向量 IW 距离的负数与阈值 $\boldsymbol{b}^1$ 之和。如果所有的阈值向量为 0，则当输入向量 $\boldsymbol{p}$ 和输入权值向量 IW 相等时，$\boldsymbol{n}^1$ 为最大值 0。

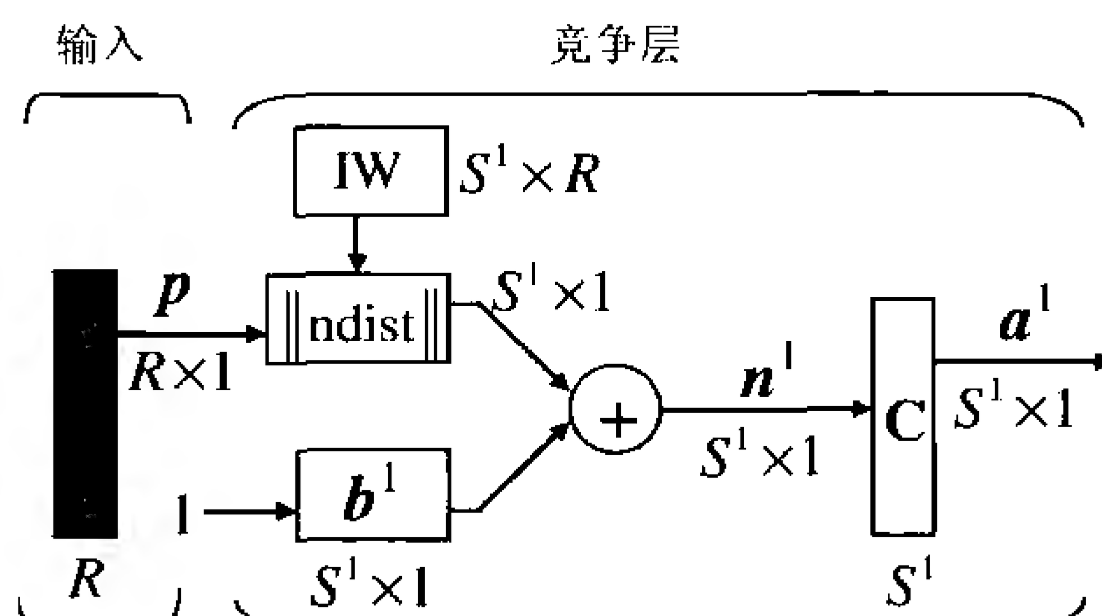


图 5-1　竞争型神经网络模型

对于 n^1 中最大的元素，竞争层传输函数输出 1（即竞争的“获胜者”输出为 1），而其他元素均输出 0。如果所有的阈值向量为 0，则当神经元的权值向量接近输入向量时，它在 n^1 各元素中的负值最小，正值最大，从而赢得竞争，对应的输出为 1。

注意

在 MATLAB 工具箱中，创建竞争型神经网络的函数是 newc。

5.1.2 竞争型神经网络的学习

1. Kohonen 权值学习规则

竞争型神经网络按 Kohonen 学习规则对获胜神经元的权值进行调整。假若第 i 个神经元获胜，则输入权值向量的第 i 行元素（即获胜神经元的各连接权）按下式进行调整

$$ {}_i\mathrm{IW}(k) = {}_i\mathrm{IW}(k-1) + \alpha[p(k) - {}_i\mathrm{IW}(k-1)] \tag{5-1} $$

而其他神经元的权值不变。

Kohonen 学习规则通过输入向量进行神经元权值的调整，因此在模式识别的应用中是很有用的。通过学习，那些最靠近输入向量的神经元权值向量得到修正，使之更靠近输入向量，其结果是获胜的神经元在一次相似的输入向量出现时，获胜的可能性会更大；而对于那些与输入向量相差很远的神经元权值向量，获胜的可能性将变得很小。这样，当经过越来越多的训练样本学习后，每一个网络层中的神经元权值向量很快被调整为最接近某一类输入向量的值。最终的结果是，如果神经元的数量足够多，则具有相似输入向量的各类模式作为输入向量时，其对应的神经元输出为 1；而对于其他模式的输入向量，其对应的神经元输出为 0。所以，竞争型网络具有对输入向量进行学习分类的能力。

注意

在 MATLAB 工具箱中，learnk 函数用于实现 Kohonen 学习规则。

2. 阈值学习规则

竞争型神经网络的一个局限性是，某些神经元可能永远也派不上用场，换句话说，某些神经元的权值向量从一开始就远离所有的输入向量，从而使得该神经元不管进行多长的训练，也永远不会赢得竞争。这些神经元称之为“死神经元”，它们实现不了任何有用的函数映射。

为了避免这一现象的发生，对那些很少获胜（甚至从来不曾获胜）的神经元赋以较大的阈值，而对那些经常获胜的神经元赋以较小的阈值。正的阈值与距离的负值相加，使获胜很少的神经元竞争层传输函数的输入就像获胜的神经元一样。这一过程就像人们同情弱者一样，表现出一个人的“良心”。

这一过程的实现，需要用到神经元输出向量的平均值，它等价于每个神经元输出为 1 的百分比，显然，经常获胜的神经元，其输出为 1 的百分比大。

注意

在 MATLAB 工具箱中，learncon 函数用于进行阈值的修正。

对学习函数 learncon 进行阈值修正时，神经元输出向量的平均值越大，其“良心”值越大，

所以凭“良心”获得的阈值就小，而让那些不经常获胜的神经元的阈值逐渐变大。其算法如下

$$c(k)=(1-\mathrm{lr})\times c(k-1)+\mathrm{lr}\times a(k-1) \tag{5-2}$$

$$b(k)=\exp[1-\log(c(k))]-b(k-1) \tag{5-3}$$

式中，c 为“良心”值；a 为神经元输出的平均值；lr 为学习率。

一般将 learncon 的学习率设置成默认值或比 learnk 的学习率小的值，使其在运行过程中能够较精确地计算神经元的输出平均值。

结果那些不经常产生响应的神经元的阈值相对于那些经常产生响应的神经元，其阈值不断增大，使其产生响应的输入空间也逐渐增大，即对更多的输入向量产生响应，最终各神经元对输入向量产生响应的数目大致相等。

这样做有两点好处。

（1）如果某个神经元因为远离所有的输入向量而始终不能在竞争中获胜，则其阈值会变得越来越大，使其终究可以获胜。当这一情况出现后，它将逐渐向输入向量的某一类聚集，一旦神经元的权值靠近输入向量的某一类模式时，该神经元将经常获胜，其阈值将逐渐减小到 0，这样就解决了“死神经元”的问题。

（2）学习函数 learncon 强迫每个神经元对输入向量的分类百分比大致相同，所以如果输入空间的某个区域比另外一个区域聚集了更多的输入向量，那么输入向量密度大致的区域将吸引更多的神经元，从而获得更细的分类。

5.1.3　竞争型神经网络存在的问题

竞争型神经网络对于模式样本本身具有较明显的分类特征，它可以对其进行正确的分类，网络对同一类或相似的输入模式具有较稳定的输出响应。但也存在一些问题。

（1）当学习模式样本本身杂乱无章，没有明显的分类特征时，网络对输入模式的响应呈现振荡的现象，即对同一类输入模式的响应可能激活不同的输出神经元，从而不能实现正确的分类。当各类模式的特征相近时，也会出现同样的情况。

（2）在权值和阈值的调整过程中，学习率的选择在收敛速度和稳定性之间存在矛盾，而不像前面介绍的其他学习算法，可以在刚开始时采用较大的学习率，而在权值和阈值趋于稳定时，采用较小的学习率。竞争型神经网络在增加新的学习样本时，对权值和阈值可能需要做比前一次更大的调整。

（3）网络的分类性能与权值和阈值的初始值、学习率、训练样本的顺序、训练时间的长短（训练次数）等都有关系，而目前还没有效的方法对各种因素的影响进行评判。

（4）在 MATLAB 神经网络工具箱中，以函数 trainr 进行竞争型神经网络的训练，用户只能限定训练的最长时间或训练的最大次数，以此终止训练，但终止训练时网络的分类性能究竟如何，没有明确的评判指标。

5.1.4　竞争型神经网络的 MATLAB 程序

竞争型神经网络适用于具有明显分类特征的模式分类。其 MATLAB 仿真程序设计主要包括如下。

（1）创建竞争型神经网络。首先根据给定的问题确定训练样本的输入向量，当不足以区分各类模式时，应想办法增加特征值；其次根据模式分类数确定神经元的数目。

（2）训练网络。训练最大次数的默认值为 100，当训练结果不能满足分类要求时，可尝试增加训练的最大次数。

（3）以测试样本进行仿真。

【例 5-1】利用竞争层网络对样本数据进行分类。

本例中待分类的样本数据由 nngenc 函数随机产生，即

P=nngenc(rand,class,num,std);

其中，参数 class 表示样本数据的类别个数。然后利用 newc 函数建立竞争层网络

net=newc(range,class,klr,clr);

其中，class 是数据类别个数，也是竞争层神经元的个数；klr 和 clr 分别是网络的权值学习速率和阈值学习速率。竞争层网络在训练时不需要目标输出，网络通过对数据分布特性的学习，自动将数据划分为指定类别数。网络的训练语句如下（其中，默认的训练函数为 trainr）

net=train(net,P);

在对训练好的网络进行仿真时，网络的输出为单值矢量组，为了观察方便，一般要将单值矢量组转化为下标矩阵的形式

Y=sim(net,P);

Y1=vec2ind(Y);

本例完整的 MATLAB 程序如下。

```
%产生样本数据 P，P 中包括三类共 30 个二维矢量
range=[-1 1;-1 1];
class=3;
num=10;
std=0.1;
P=nngenc(range,class,num,std);
%画出样本数据分布图
plot(P(1,:),P(2,:),'*','markersize',6);axis([-1.5 1.5 -1.5 1.5]);
%建立竞争层网络
klr=0.1;
clr=0.01;
net=newc(range,class,klr,clr);
%对网络进行训练
net.trainParam.epochs=5;
net=train(net,P);
%画出竞争层神经元权值
w=net.iw{1};
hold on;
plot(w(:,1),w(:,2),'ob');
title('Input data & Weights');
%利用原始样本数据对网络进行仿真
Y=sim(net,P);
Y1=vec2ind(Y) ;
%用不同符号标出数据分类结果
figure;
```

```
for i=1:30
    if Y1(i)==1
        plot(P(1,i),P(2,i),'*','markersize',6);
    elseif Y1(i)==2
        plot(P(1,i),P(2,i),'+','markersize',6);
    else
        plot(P(1,i),P(2,i),'x','markersize',6);
    end
    hold on;
end
axis([-1.5 1.5 -1.5 1.5]);
title('class 1:*    class 2:+    class 3:x');
%利用一组新的输入数据检验网络性能
p=[-0.4 -0.4;-0.1 0.9];
y=sim(net,p);
y1=vec2ind(y)
```

窗口显示结果为

```
TRAINR, Epoch 0/5
TRAINR, Epoch 5/5
TRAINR, Maximum epoch reached.
y1 =
    1     2
```

程序运行结果如图 5-2 和图 5-3 所示。在图 5-2 中，待分类的样本数据用星号标志，网络训练完毕后的竞争层神经元权值由圆圈标注。在图 5-3 中，已经划分好的三类数据分别用星号、加号和“×”符号标注。

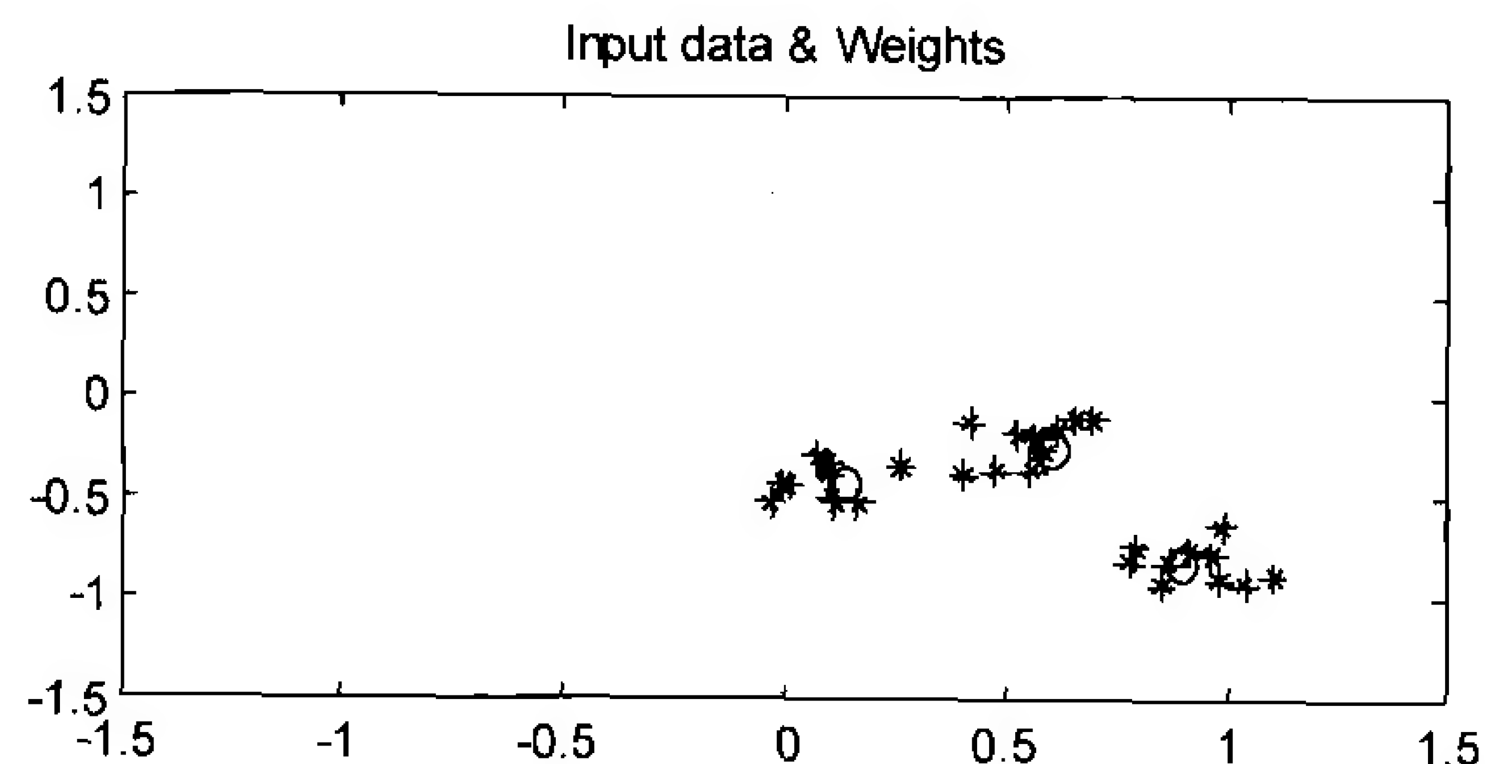


图 5-2　待分类的样本数据和权值

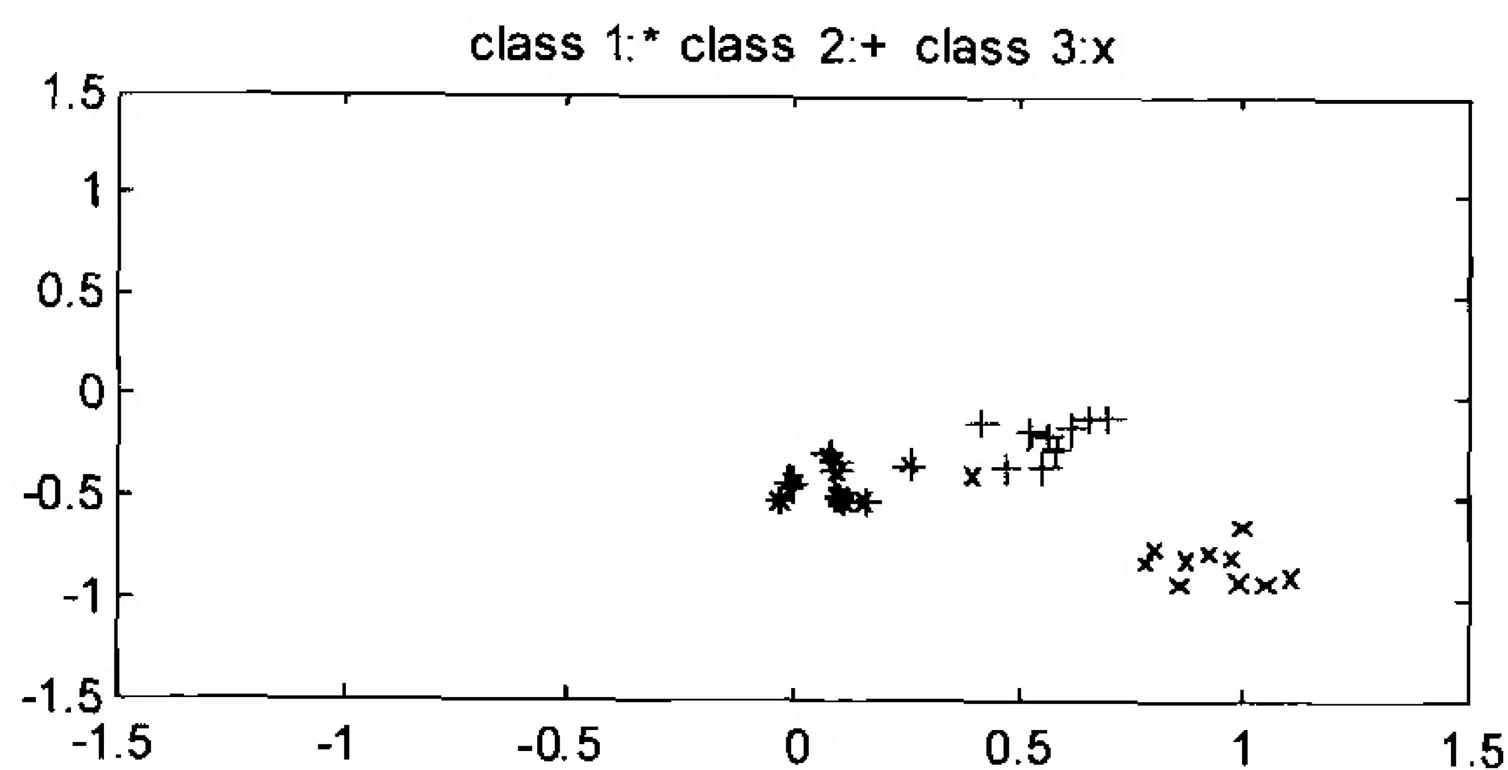


图 5-3　网络分类结果

【例 5-2】以竞争型神经网络完成如图 5-4 所示的三类模式的分类。

$$p_1=\begin{bmatrix}-0.1961\\0.9806\end{bmatrix},p_2=\begin{bmatrix}0.1961\\0.9806\end{bmatrix},p_3=\begin{bmatrix}0.9806\\0.1961\end{bmatrix}$$

$$p_4=\begin{bmatrix}0.9806\\-0.1961\end{bmatrix},p_5=\begin{bmatrix}-0.5812\\-0.8137\end{bmatrix},p_6=\begin{bmatrix}-0.8137\\-0.5812\end{bmatrix}$$

图 5-4 中的三类模式从它们在二维平面上的位置特征看，具有明显的分类特征，可以以竞争型神经网络完成其分类。

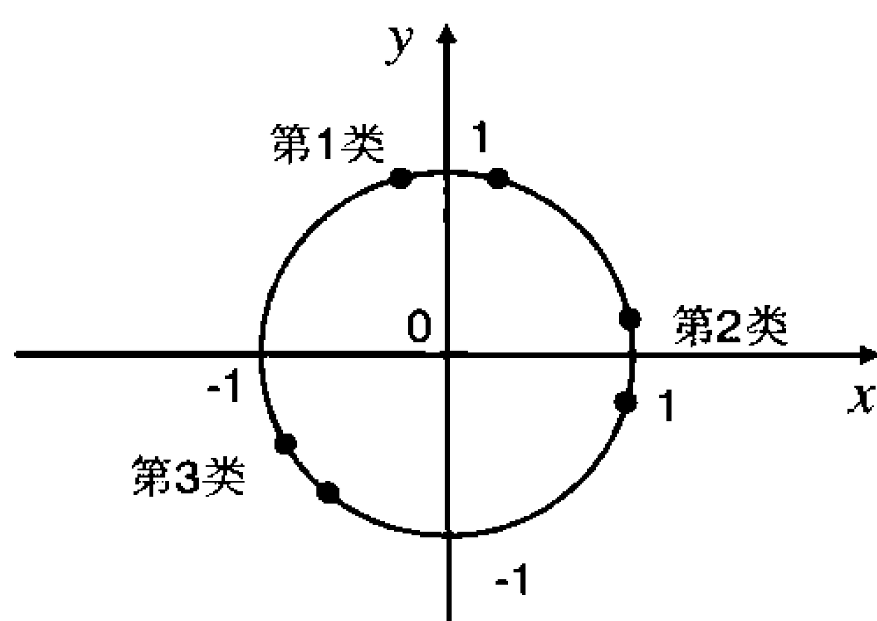


图 5-4　待分类模式

本例的 MATLAB 程序如下。

```
clear all
%定义输入向量
P=[-0.1961 0.1961 0.9806 0.9806 -0.5812 -0.8137;
     0.9806 0.9806 0.1961 -0.1961 -0.8137 -0.5812];
%创建竞争型网络
net=newc([-1 1;-1 1],3);
%训练神经网络
net=train(net,P);
%定义待测试样本输入向量
p=[-0.1961 0.1961 0.9806 0.9806 -0.5812 -0.8137;
     0.9806 0.9806 0.1961 -0.1961 -0.8137 -0.5812];
%网络仿真
y=sim(net,p);
%输出仿真结果
yc=vec2ind(y)
```

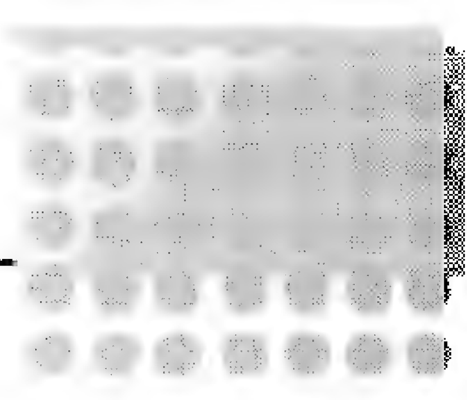
仿真结果为

```
yc =
     2     2     1     1     3     3
```

结果表明很好地完成了分类。

5.2　自组织特征映射神经网络

自组织特征映射网络也称为 Kohonen 网络，或者称为 Self-Organizing Feature Map（SOM）网络，它是由芬兰学者 Teuvo Kohonen 于 1981 年提出的。该网络是一个由全连接的神经元阵列组成的无教师自组织、自学习网络。Kohonen 认为，处于空间中不同区域的神经元有不同的分工，当一个神经网络接受外界输入模式时，将会分为不同的反应区域，各区域对输入模式具有不同的响应特性。



5.2.1 特征映射网络的模型

特征映射网络结构如图 5-5 所示。

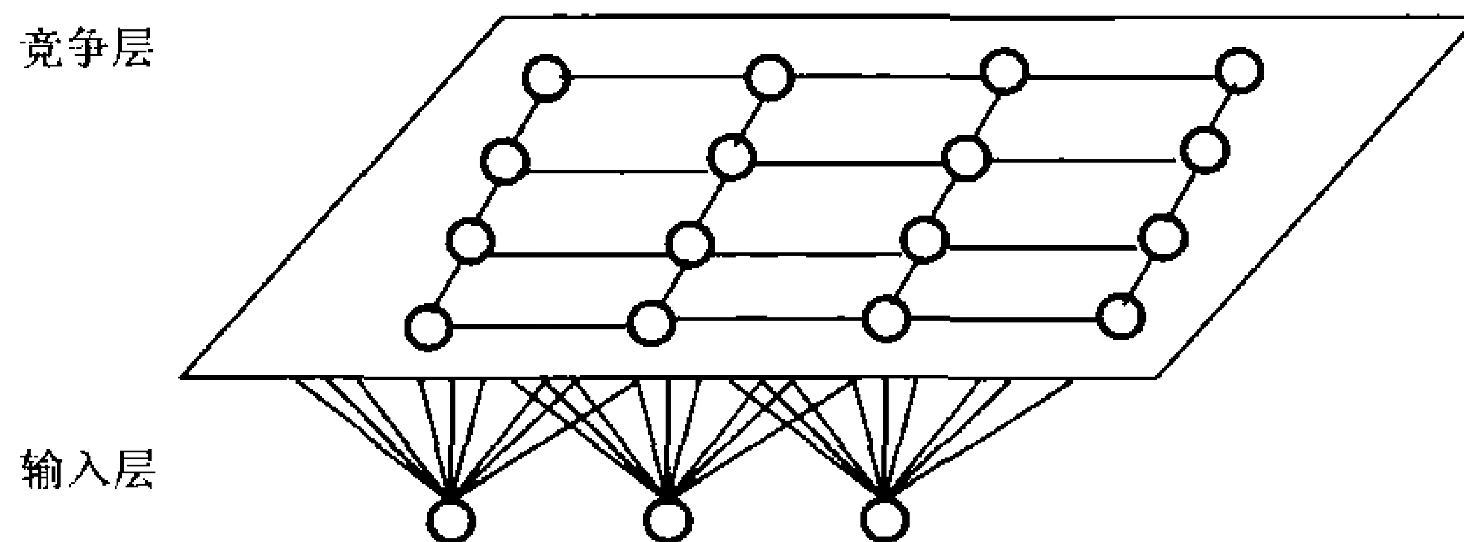


图 5-5 特征映射网络结构

特征映射网络的一个典型特性就是可以在一维或二维的处理单元阵列上，形成输入信号的特征拓扑分布，因此特征映射网络具有抽取输入信号模式特征的能力。特征映射网络一般只包含有一维阵列和二维阵列，但也可以推广到多维处理单元阵列中去。下面只讨论应用较多的二维阵列。特征映射网络模型由以下 4 部分组成。

- 处理单元阵列。用于接收事件输入，并且形成对这些信号的“判别函数”。
- 比较选择机制。用于比较“判别函数”，并选择一个具有最大函数输出值的处理单元。
- 局部互连作用。用于同时激励被选择的处理单元及其最邻近的处理单元。
- 自适应过程。用于修正被激励的处理单元的参数，以增加其对应于特定输入“判别函数”的输出值。

假定网络输入为 $\boldsymbol{X} \in R^n$，输出神经元 i 与输入单元的连接权值为 $\boldsymbol{W}_i \in R^n$，则输出神经元 i 的输出为 o_i 为

$$o_i = \boldsymbol{W}_i \boldsymbol{X} \tag{5-4}$$

网络实际具有响应的输出单元为 k，该神经元的确定是通过“赢者通吃”的竞争机制得到的，其输出为

$$o_k = \max_i \{o_i\} \tag{5-5}$$

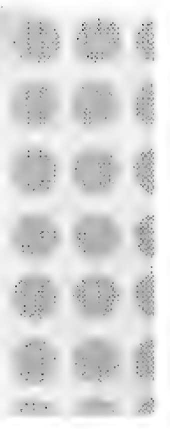
以上两式可修正为

$$o_i = \sigma\left(\varphi_i + \sum_{t \in S_t} r_k o_t\right), \quad \varphi_i = \sum_{j=1}^{m} w_{ij} x_j, \quad o_k = \max_i \{o_i\} - \varepsilon$$

其中，w_{ij} 为输出神经元 i 和输入神经元 j 之间的连接权值。x_j 为输入神经元 j 的输出。$\sigma(t)$ 为非线性函数，即

$$\sigma(t) = \begin{cases} 0 & t < 0 \\ \sigma(t) & 0 \leqslant t \leqslant A \\ A & t > A \end{cases} \tag{5-6}$$

ε 为一个很小的正数，r_k 为系数，它与权值及横向连接有关。S_i 为与处理单元 i 相关的处理单元集合，o_k 称为浮动阈值函数。



5.2.2 特征映射网络的学习

特征映射的学习算法过程如下。

（1）初始化。对 N 个输入神经元到输出神经元的连接权值赋予较小的权值。选取输出神经元 j 个“邻接神经元”的集合 S_j。其中，$S_j(0)$ 表示时刻 $t=0$ 的神经元 j 的“邻接神经元”的集合，$S_j(t)$ 表示时刻 t 的“邻接神经元”的集合。区域 $S_j(t)$ 随着时间的增长而不断缩小。

（2）提供新的输入模式 $\boldsymbol{X}$。

（3）计算欧氏距离 d_j，即输入样本与每个输出神经元 j 之间的距离。

$$d_j = \left\|\boldsymbol{X} - \boldsymbol{W}_j\right\| = \sqrt{\sum_{i=1}^{N}\left[x_i(t) - w_{ij}(t)\right]^2} \tag{5-7}$$

并计算出一个具有最小距离的神经元 j^*，即确定出某个单元 k，使得对于任意的 j，都有 $d_k = \min_j(d_j)$。

（4）给出一个周围的领域 $S_k(t)$。

（5）按照下式修正输出神经元 j^* 及其“邻接神经元”的权值。

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)\left[x_i(t) - w_{ij}(t)\right]$$

其中，η 为一个增益项，并随时间变化逐渐下降到零，一般取

$$\eta(t) = \frac{1}{t} \text{或} \eta(t) = 0.2\left(1 - \frac{t}{10000}\right)$$

（6）计算输出 o_k。

$$o_k = f\left(\min_j\left\|\boldsymbol{X} - \boldsymbol{W}_j\right\|\right)$$

其中，$f(\cdot)$ 一般为 0—1 函数或其他非线性函数。

（7）提供新的学习样本来重复上述学习过程。

5.2.3 基于特征映射网络的人口分类

【例 5-3】

人口分类是人口统计中的一个重要指标。由于各方面的原因，我国人口的出生率在性别上的差异比较大，具体表现在同一个时期出生的人口中，一般男的占多数，大大超过了正常的比例。因此，正确地进行人口分类是制定合理的人口政策的基础。

1. 样本设计

通过分析历史资料，得到了在 1999 年 12 月共 20 个地区的人口出生比例情况，如表 5-1 所示。

表 5-1 人口出生比例

男 (%)	0.5512	0.5123	0.5087	0.5001	0.6012	0.5298	0.5000	0.4965	0.5103	0.5003
女 (%)	0.4488	0.4877	0.4913	0.4999	0.3988	0.4702	0.5000	0.5035	0.4897	0.4997

将上表中的数据作为网络的输入样本 ***P***。***P*** 是一个二维随机向量，它的分布情况如图 5-6 所示。

```
P=[0.5512 0.5123   0.5087   0.5001   0.6012   0.5298   0.5000   0.4965   0.5103
  0.5003; 0.4488   0.4877   0.4913   0.4999   0.3988   0.4702   0.5000   0.5035
  0.4897  0.4997];
  plot(P(1,:),P(2,:),'*r');
  hold on
```

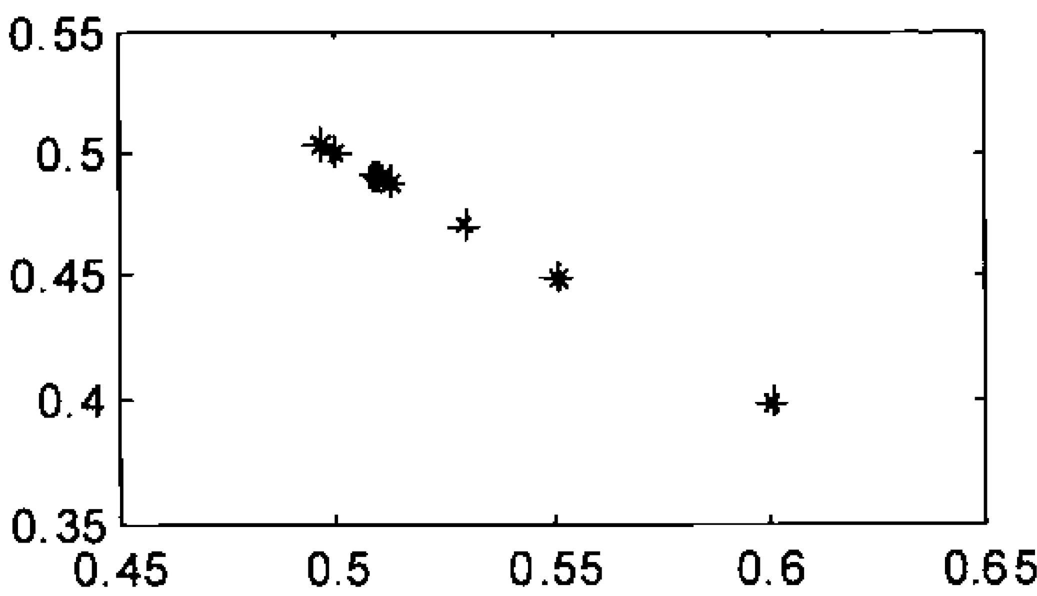


图 5-6　样本数据的分布

2．网络创建

利用 12 个神经元的特征映射网络对输入向量 ***P*** 进行分类。该网络竞争层神经元的组织结构为 3×4，通过距离函数 linkdist 来计算距离。网络创建代码如下。

```
net=newsom([0 1;0 1],[3 4]);
w1_init=net.iw{1,1};
plotsom(w1_init,net.layers{1}.distances);
```

运行结果如图 5-7 所示，图中每一点表示一个神经元，由于网络的初始权值都被设置为 0.5，所以这些点在图中是重合的，看起来就像一个点，实际上是 12 个点。

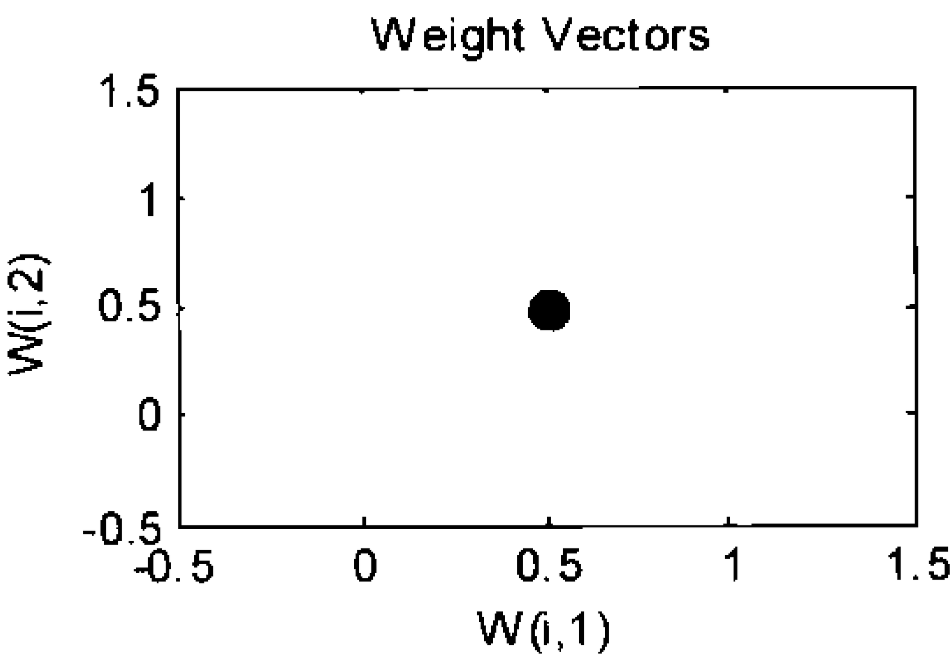


图 5-7　网络初始权值的分布

在命令窗口中查看 w1_init 的值，可得

```
w1_init =
      0.5000      0.5000
      0.5000      0.5000
      0.5000      0.5000
      0.5000      0.5000
      0.5000      0.5000
```

```
0.5000    0.5000
0.5000    0.5000
0.5000    0.5000
0.5000    0.5000
0.5000    0.5000
0.5000    0.5000
0.5000    0.5000
```

3. 网络训练与测试

接下来利用训练函数 train 对网络进行训练，设想经过训练的网络可对输入向量进行正确分类。网络训练步数对于网络性能的影响比较大，所以这里将步数设置为 100、300 和 500，并分别观察其权值分布。

步数为 100 时的权值分布如图 5-8 所示。

训练步数为 100 时的训练代码如下。

```
net=train(net,P);
figure;
w1=net.iw{1,1};
plotsom(w1,net.layers{1}.distances)
```

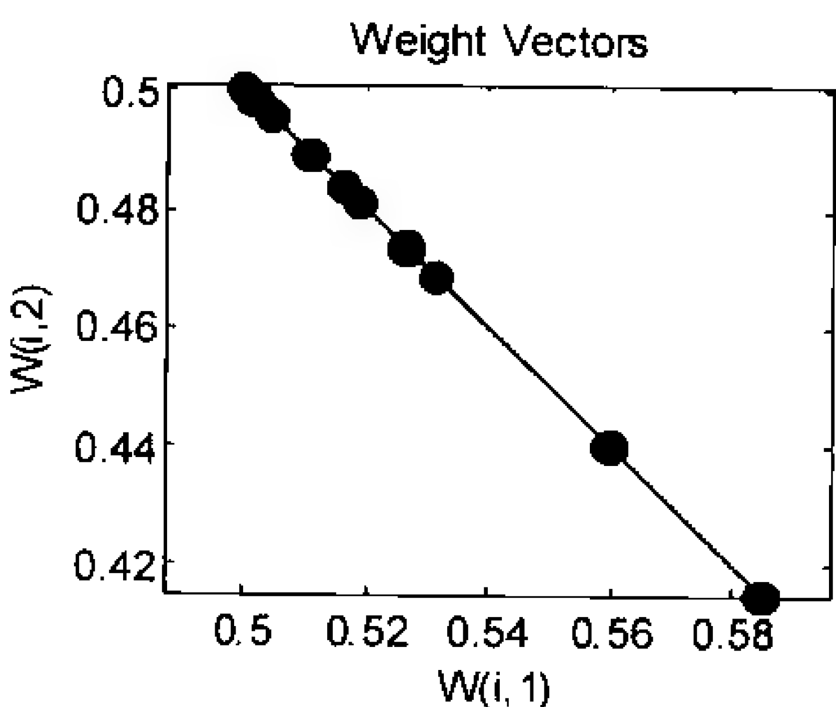


图 5-8 权值分布(训练步数为 100)

步数为 300 时的权值分布如图 5-9 所示。

```
%训练步数为 300 时的训练代码
net.trainParam.epochs=300;
net=init(net);
net=train(net,P);
figure;
w1=net.iw{1,1};
plotsom(w1,net.layers{1}.distances)
```

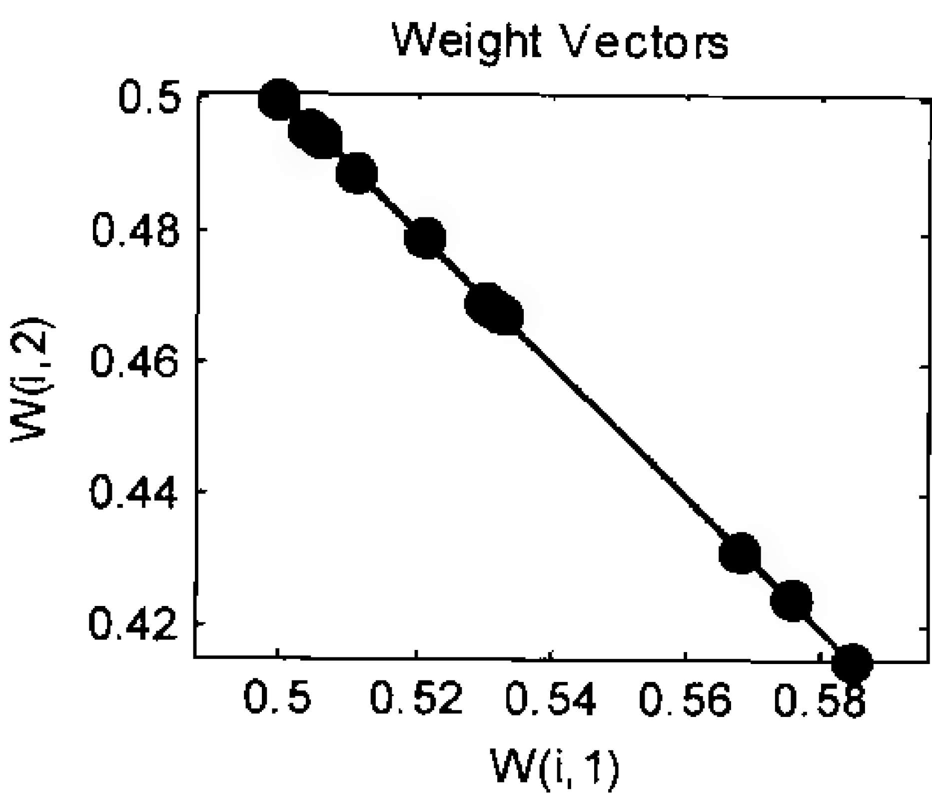


图 5-9 权值分布(训练步数为 300)

步数为 500 时的权值分布如图 5-10 所示。

```
%训练步数为 500 时的训练代码
net.trainParam.epochs=500;
net=init(net);
net=train(net,P);
figure;
w1=net.iw{1,1};
plotsom(w1,net.layers{1}.distances)
```

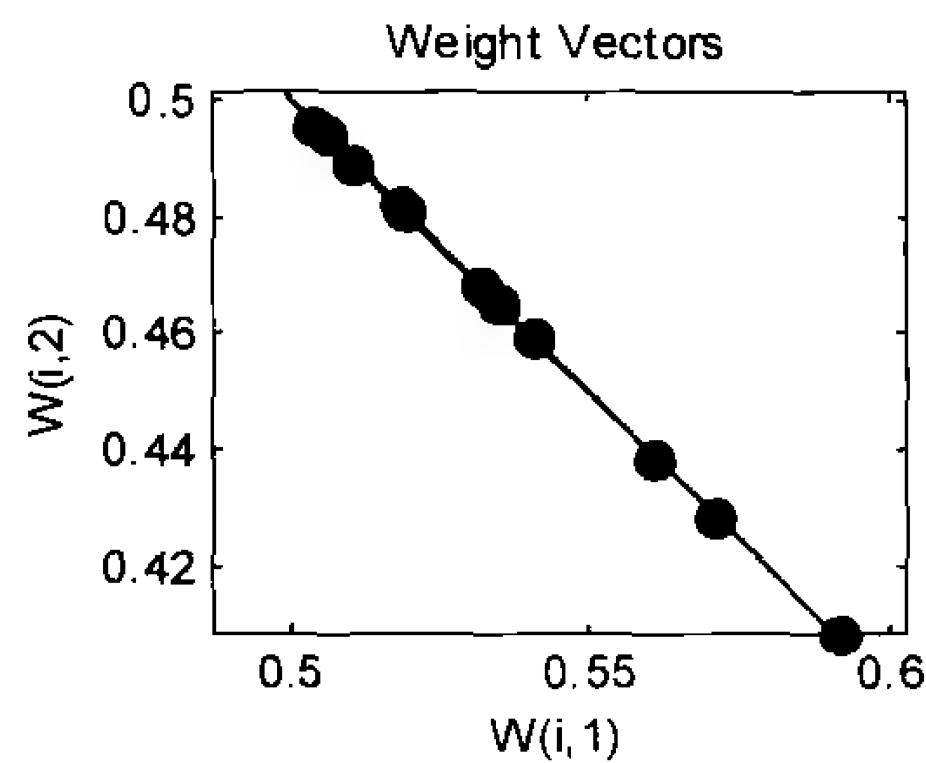


图 5-10　权值分布(训练步数为 500)

从图 5-8、图 5-9 和图 5-10 可以看出，训练了 100 步以后，神经元就开始自组织地分布了，每个神经元可以区分不同的样本。随着训练步数的增多，神经元的分布更加合理，但是，当训练次数达到一定值后，权值分布的改变就不明显了。比如，训练 300 步和训练 500 步后的权值分布就比较相似。

网络训练结束后，权值也就固定了。以后每输入一个值，网络就会自动地对其进行分类。因此，利用这一点对网络进行测试。首先，利用仿真函数 sim 来观察网络对样本数据的分类结果。

```
y=sim(net,P);
Y=vec2ind(y)
```

输出结果为

```
Y =
     4    10    10    12     1     6    12    12    10    12
```

对结果进行分析，如表 5-2 所示。

表 5-2　聚类结果

样本序号	类　别	激发神经元的索引
1	1	4
2	2	3
3	3	10
4，7，10	4	11
5	5	1
6	6	7
8	7	12
9	8	6

现在，输入一个某地的出生性别比例，检验它属于哪一类。

```
p=[0.5;0.5];
y=sim(net,p);
y=vec2ind(y)
```

结果为 11。由此可见，此时激发了网络的第 11 个神经元，所以 **p** 属于第 4 类。通过直接对比数据可知，**p** 确定与样本中的第 4 组、第 7 组和第 10 组数据非常接近。

5.3 自适应共振理论

自适应共振理论英文全称为 Adaptive Resonance Theory，简称 ART，它是由 S.Grossberg 和 A.Carpentent 等人于 1986 年提出的。Grossberg 的研究工作主要是采用数学方法描述人的心理和认知活动，致力于为人类的心理和认知活动建立一个统一的数学模型。以其思想基础提出的 ART 模型成功地解决了神经网络学习中的稳定性（固定某一分类集）和可塑性（调整网络固有参数的学习状态）的关系问题。

ART 是以认知和行为模式为基础的一种无教师、矢量聚类和竞争学习的算法。在数学上，ART 为非线性微分方程的形式；在网络结构上，ART 是全反馈结构，且各层节点有不同的性质。

ART 网络共有 3 种类型：ART-1、ART-2 和 ART-3。这里主要介绍第一类网络。

5.3.1 自适应共振理论模型

ART-1 型网络结构如图 5-11 所示。由图可见，网络分为输入和输出两层，一般根据各层所有的功能特征称输入层为比较层，输出层为识别层。和其他阶层型网络的显著区别是，ATR-1 型网络不仅具有从输入层到输出层的前馈连接权，还有从输出层到输入层的反馈连接权。

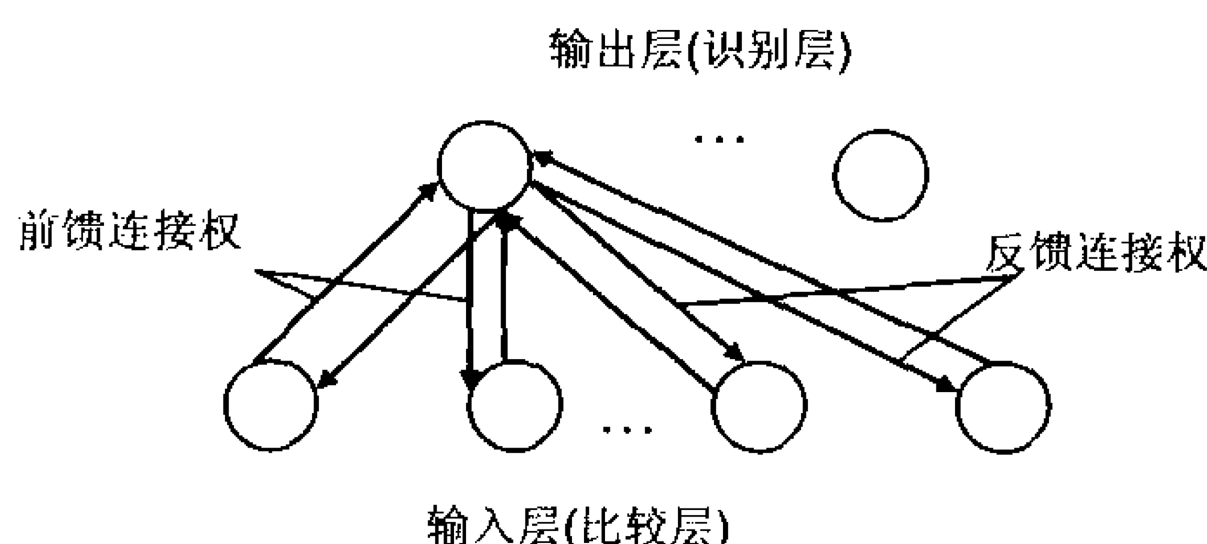


图 5-11 ART-1 型网络结构

假定网络输入层有 N 个神经元，输出层有 M 个神经元，二值输入模式和输出向量分别为 $\boldsymbol{A}_k=\left(a_1^k,a_2^k,\cdots,a_N^k\right)$ 和 $\boldsymbol{B}_k=\left(b_1^k,b_2^k,\cdots,b_M^k\right)$，其中 $k=1,2,\cdots,p$。p 为输入学习模式的数目。前馈连接权和反馈连接权分别为 w_{ij} 和 t_{ij}，$j=1,2,\cdots,M$。

ART-1 网络的学习及工作过程，是通过反复将输入学习模式由输入层向输出层自下而上地识别、由输出层向输入层自上而下地比较来实现的。当这种自下而上的识别和自上而下的比较达到共振，即输入向量可以正确反映输入学习模式的分类，且网络原有记忆没有受到不良影响时，网络对一个输入学习模式的记忆和分类就算完成。网络的学习和工作过程可以分为初始化阶段、识别阶段、比较阶段和探寻阶段，下面将详细介绍网络的学习及工作过程。

5.3.2 自适应共振理论的学习

ART-1 网络的学习及工作可以归纳为如下过程。

（1）初始化。令 $t_{ij}(0)=1$，$w_{ij}(0)=\dfrac{1}{N+1}$，$i=1,2,\cdots,N$，$j=1,2,\cdots,M$。其中，警戒参数 $0<\rho\leqslant 1$。

（2）将输入模式 $\boldsymbol{A}_k=\left(a_1^k,a_2^k,\cdots,a_N^k\right)$ 提供给网络的输入层。

（3）计算输出层各个神经元的输入加权和。

$$s_j=\sum_{i=1}^{N}w_{ij}a_i^k\,,\quad j=1,2,\cdots,M$$

（4）选择输入模式的最佳分类结果。

$$s_g=\max_{j=1,2,\cdots,M}(s_j)$$

令神经元 g 的输出为 t。

（5）计算以下 3 式，并进行判断。

$$|\boldsymbol{A}_k|=\sum_{i=1}^{N}a_i^k$$

$$|\boldsymbol{T}_g\cdot\boldsymbol{A}_k|=\sum_{i=1}^{N}t_{gi}a_i^k$$

$$\frac{|\boldsymbol{T}_g\cdot\boldsymbol{A}_k|}{|\boldsymbol{A}_k|}>\rho$$

如果最后一式成立，则转入步骤（7），否则转入步骤（6）。

（6）取消识别结果，将输出层神经元 g 的输出值复位为 0，并将这一神经元排除在下次识别的范围之外，返回步骤（4）。当所有已利用过的神经元都无法满足步骤（5）中的最后一式时，则选择一个新的神经元作为分类结果，并进入步骤（7）。

（7）接受识别结果，并按照下式调整连接权值。

$$w_{ig}(t+1)=\frac{t_{gi}(t)a_i}{0.5+\sum_{i=1}^{N}t_{gi}(t)a_i}$$

$$t_{gi}(t+1)=t_{gi}(t)a_i$$

其中，$i=1,2,\cdots,N$。

（8）将步骤（6）中复位的所有神经元重新加入识别范围中，返回步骤（2），对下一个模式进行识别。

无论网络学习还是回想，都使用以上规则。只不过在网络回想时，只对那些与未使用过的输出神经元有关的连接权值 w_{ij} 和 t_{ij} 进行初始化。其他连接权值保持网络学习后的值不变。当

输入模式是一个网络已记忆的学习模式时，不需要再进行网络权值的调整，这是因为当输入模式和网络记忆的学习模式完全相等时，再按照权值调整公式进行调整，网络连接权值不会发生任何变化。而当输入模式与网络记忆模式存在一定差异时，按照权值调整公式进行调整，将会影响网络原有模式的记忆效果。但是如果输入的是一个全新的模式，需要利用网络对其另外记忆时，则必须按照权值调整公式对网络连接权值进行调整。

尽管 ART-1 网络具有许多其他网络所没有的优点，但是它仅以输出层中某个神经元代表分类结果，而不是像 Hopfield 网络那样，把分类结果分散在各个神经元上来表示。所以，一旦输出层中某个输出神经元损坏，则会导致该神经元所代表类别的模式信息全部消失。这是 ART-1 网络一个很大的缺陷。

ART-2 型网络与 ART-1 型的主要区别是，ART-2 型网络以模拟量作为输入模式，同时在算法上做了一些相应的改进，并采用慢速学习方式，其抗干扰能力大大增强。ART-3 型网络是由多个 ART-1 型网络组成的复合阶层型网络。

5.3.3 自适应共振理论的 MATLAB 程序

MATLAB 神经网络工具箱没有为 ART 型网络提供专门的函数，因此，利用现有的神经网络工具箱是无法实现 ART-1 网络的。但是，我们可以借助于 MATLAB 强大的数学计算功能来实现 ART-1 网络的训练和联想记忆功能。

【例 5-4】现举一个简单的例子来演示利用 MATLAB 实现 ART-1 网络的过程。如图 5-12 所示，设 ART-1 网络有 5 个输入神经元和 20 个输出神经元。现有两组输入模式 $\boldsymbol{A}_1=(1,1,0,0,0)$和 $\boldsymbol{A}_2=(1,0,0,0,1)$，要求利用这两组模式来训练网络。根据 5.3.2 节中的训练过程，该网络的训练步骤分为下面几步。

（1）初始化。令 $w_{ij}=1/n+1=1/6$，$t_{ji}=1$，其中 $i=1,2,\cdots,5$，$j=1,2,\cdots,20$，令 $\rho=0.8$。

（2）将输入模式 $\boldsymbol{A}_1$ 提供给网络的输入层。

（3）求获胜的神经元。因为在网络的初始状态下，所有的前馈连接权 w_{ij} 均取相等的权值 $1/6$，所以各输入神经元均具有相同的输入加权和 s_j。这时可取任意一个神经元作为 $\boldsymbol{A}_1$ 的分类代表，如第 1 个，令其输出值为 1。

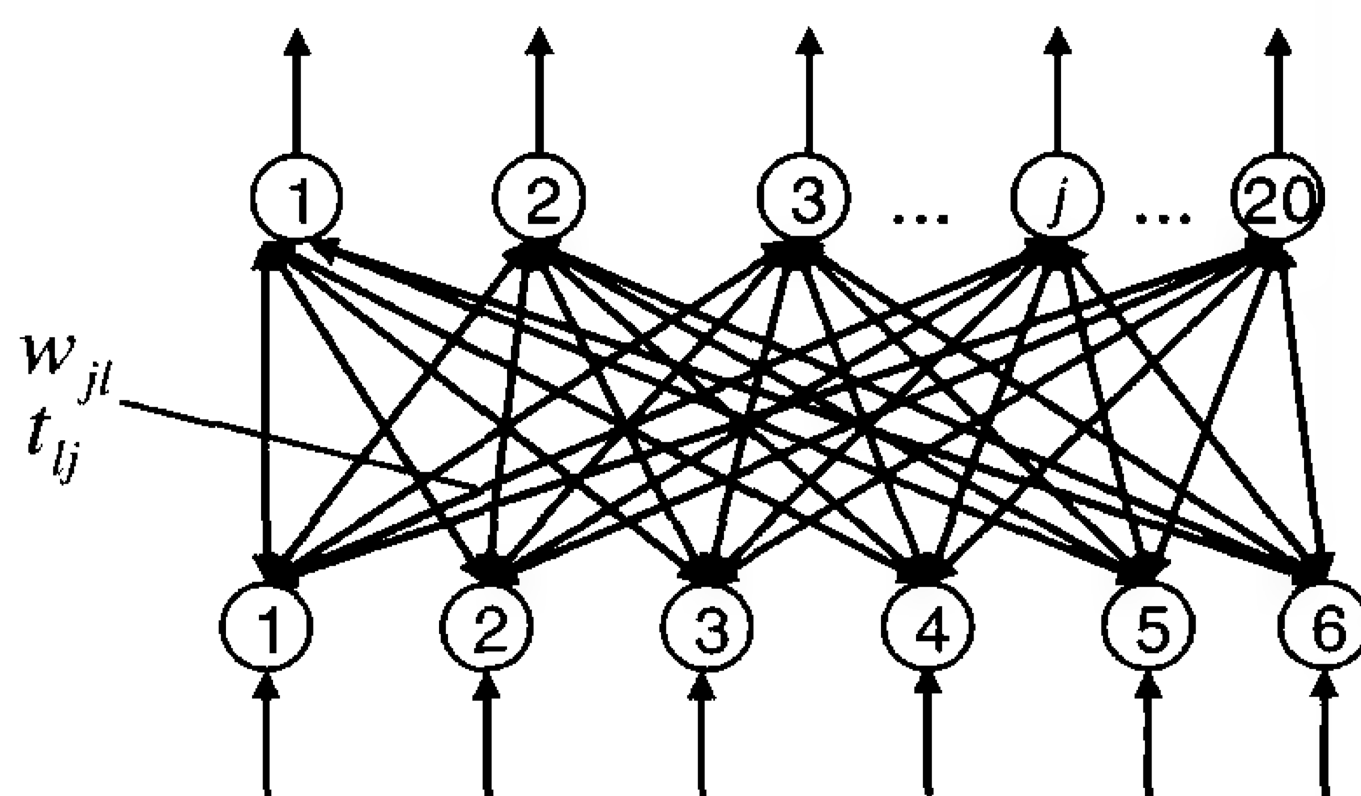


图 5-12　ART-1 网络实例

（4）计算下式。

$$|\boldsymbol{A}_1|=\sum_{i=1}^{5}a_i=2\,,\quad |\boldsymbol{T}_1\boldsymbol{A}_1|=\sum_{i=1}^{5}t_{1i}a_i=2$$

（5）计算 $\frac{|\boldsymbol{T}_1\boldsymbol{A}_1|}{|\boldsymbol{A}_1|}=1>0.8$，接受这次识别结果。

（6）调整权值。

$$\boldsymbol{W}_1=(w_{11},w_{12},w_{13},w_{14},w_{15})=(0.4,0.4,0,0,0)$$
$$\boldsymbol{T}_1=(t_{11},t_{21},t_{31},t_{41},t_{51})=(1,1,0,0,0)$$

至此，$\boldsymbol{A}_1$ 已经被记忆在网络中了。

（7）将输入模式 $\boldsymbol{A}_2$ 提供给网络的输入层。

（8）求获胜神经元，$s_1=0.4$，s_2=1/6，$s_3=\cdots=s_{20}=1/6$，由于 $s_1>s_2=s_3=\cdots=s_{20}$，所以取神经元 1 作为获胜神经元，但这显然与 $\boldsymbol{A}_1$ 的识别结果相矛盾。又因为

$$\frac{|\boldsymbol{T}_2\boldsymbol{A}_2|}{|\boldsymbol{A}_2|}=\frac{1}{2}<0.8$$

所以拒绝这次识别结果，重新进行识别。由于 $s_2=s_3=\cdots=s_{20}=1/6$，故可从中任选一个神经元作为 $\boldsymbol{A}_2$ 的分类结果，如神经元 20。

（9）调整权值。

$$\boldsymbol{W}_2=(w_{21},w_{22},w_{23},w_{24},w_{25})=(0.4,0,0,0,0.4)$$
$$\boldsymbol{T}_2=(t_{12},t_{22},t_{32},t_{42},t_{52})=(1,0,0,0,1)$$

至此，$\boldsymbol{A}_2$ 也记忆在网络中了。

按照上述步骤，可以编写以下 MATLAB 代码。

```
%竞争层的输出
xiu=rands(20);
%正向权值 W 和反向权值 T
W=rands(20,5);
T=rands(20,5);
%警戒参数
xiuxiu=0.8;
%两组模式 A1 和 A2
A1=[1 1 0 0 0];
A2=[1 0 0 0 1];
%初始化
for i=1:20
    for j=1:5
        W(i,j)=1/6;
        T(i,j)=1;
    end
end
%判定是否接受识别结果
normalA1=norm(A1,1);
normalTA1=T(1,:)*A1';
count=1;
if normalTA1/normalA1>xiuxiu
```

```
    xiu(count)=1;
end
%权值调整
W(1,:)=[0.4 0.4 0 0 0];
T(1,:)=[1 1 0 0 0];
%寻找可以记忆 A2 的神经元
for k=1:20
    s(k)=W(k,:)*A2';
    if s(k)==max(s)
        count=k;
    end
end
%如果和 A1 的神经元重复，继续寻找
if xiu(count)==1
    newcount=count+1
end
for i=1:(count-1)
    p(i)=s(i);
end
for i=count:19
    p(i)=s(i+1);
end
for k=newcount:20
    if s(k)==max(p)
        count=k;
    end
end
%确定找到的神经元序号 count，并令其对应的输出为 1
xiu(count)=1;
%权值调整
W(count,:)=[0.4,0,0,0,0.4];
T(count,:)=[1,0,0,0,1];
xiu'
```

运行结果为

```
xiu' =
1.0000    0.6375   -0.1397    0.7806    0.4698    0.3746   -0.3078   -0.6679   -0.6888
-0.6178   -0.1551    0.7120   -0.0195    0.6319   -0.0785   -0.0853   -0.0986   -0.1756
0.8032    1.0000
```

可见，第 1 个和第 20 个神经元的输出均为 1，表示它们记忆了输入模式。

5.4 学习矢量量化神经网络

5.4.1 学习矢量量化的神经网络模型

因为对于学习矢量量化（LVQ）网络，用户指定了目标分类结果，所以网络可以通过监督学习完成对输入矢量模式的准确分类。LVQ 神经网络模型如图 5-13 所示。

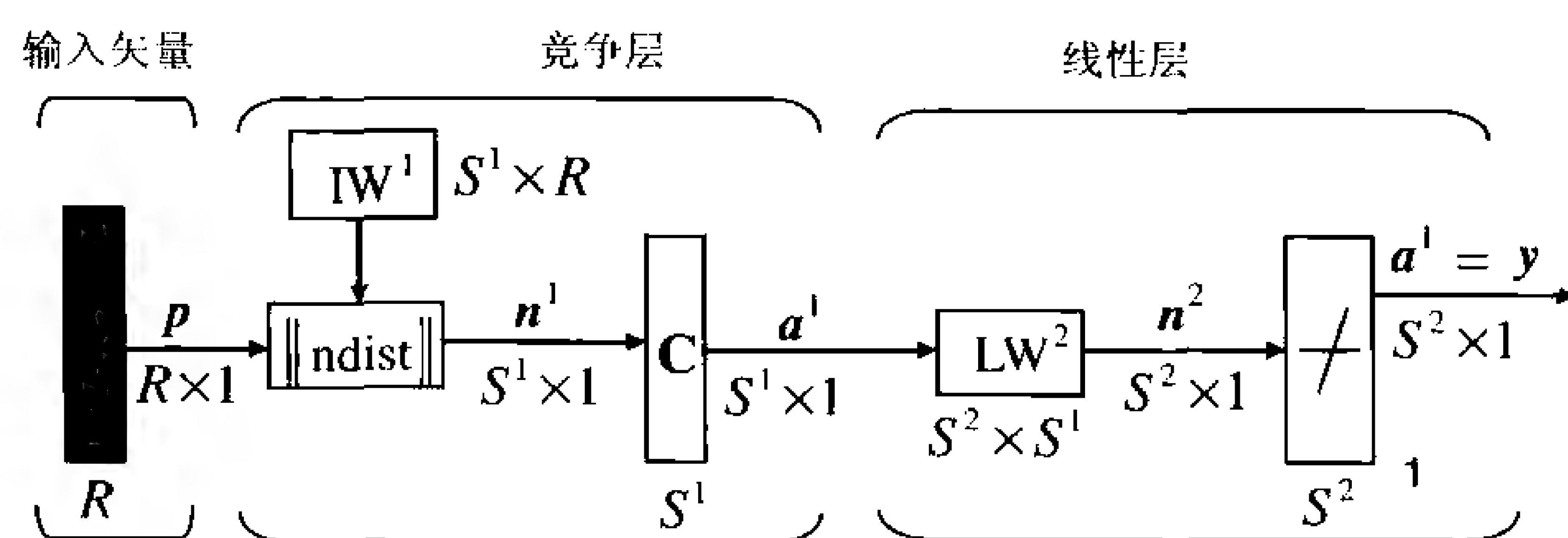
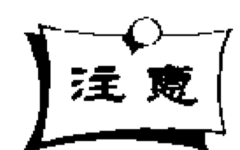


图 5-13 LVQ 神经网络模型

LVQ 神经网络有两个网络层，即竞争层和线性层。竞争层对输入矢量的学习分类与前面所阐述的竞争层一样，我们把竞争层的分类称为子分类；线性层根据用户的要求将竞争层的分类结果映射到目标分类结果中，我们把线性层的分类称为目标分类。

竞争层和线性层的每一个神经元的输出都对应一个分类（子分类或目标分类）结果，所以竞争层通过学习，可以得到 S^1 类子分类结果；然后，线性层将 S^1 类子分类结果再分成 S^2 类目标分类结果（S^1 始终大于 S^2）。例如，假设竞争层的第 1、2、3 个神经元对输入空间的子分类所对应的线性层的目标分类为第 2 类，则竞争层的第 1、2、3 个神经元与线性层的第 2 个神经元的连接权将全部为 1，而与其他线性层神经元的连接权全部为 0，这样，当竞争层的第 1、2、3 个神经元中的任意一个神经元在竞争中获胜时，线性层的第 2 个神经元将输出 1。



在 MATLAB 神经网络工具箱中，创建 LVQ 网络的函数为 newlvq。

5.4.2 学习矢量量化神经网络的学习

学习矢量量化（LVQ1）神经网络的学习与其他有导师学习方法一样，其训练样本集的输入向量和目标向量是成对出现的，即

$$\{\boldsymbol{p}_1,\boldsymbol{t}_1\},\{\boldsymbol{p}_2,\boldsymbol{t}_2\},\cdots,\{\boldsymbol{p}_Q,\boldsymbol{t}_Q\}$$

每个目标向量，除了有一个元素为 1 以外，其余元素均为 0，目标向量中元素为 1 的行即为相应的输入矢量模式。例如，对具有 3 个输入元素、4 个输出模式的 LVQ 网络

$$\left\{\boldsymbol{p}_1=\begin{bmatrix}1\\-2\\0\end{bmatrix},\boldsymbol{t}_1=\begin{bmatrix}0\\1\\0\\0\end{bmatrix}\right\}$$

则表示第 1 个训练样本对应于第 2 个模式，LVQ 网络输出层的第 2 个神经元输出 1。

LVQ1 网络进行训练时，对每一个输入矢量 $\boldsymbol{p}$，先以函数 ndist 计算它与输入权值向量 IW^1 每一行元素的距离，使隐层神经元进行竞争。假设 $\boldsymbol{n}^1$ 的第 i 个元素值最大，则竞争层的第 i 个神经元将赢得竞争，这使得竞争层的输出 $\boldsymbol{a}^1$ 的第 i 个元素值为 1，而其余元素为 0。

当 $\boldsymbol{a}^1$ 与第二网络层的权值 LW^2 相乘时，在 $\boldsymbol{a}^1$ 中唯一一个值为 1 的元素被认为是与输入矢量对应的第 k 个分类模式，所以网络认为输入矢量 $\boldsymbol{p}$ 为 k 个分类模式，则 $\boldsymbol{a}^2$ 的第 k 个元素输出 1。当然，该分类结果可能正确，也可能不正确，因为目标向量 $\boldsymbol{t}_k$ 可能为 1，也可能为 0，它取决于输入矢量模式是否属于第 k 个分类模式。

可以根据目标矢量，调整IW^1的第i行，当分类结果正确时，使该行元素的值向输入矢量$\boldsymbol{p}$靠拢；当分类结果错误时，使该行元素的值远离输入矢量$\boldsymbol{p}$。当输入矢量$\boldsymbol{p}$得到正确的分类时

$$a_k^2 = t_k = 1 \tag{5-8}$$

IW^1的第i行可以按下式进行修正

$${}_iIW^1(q) = {}_iIW^1(q-1) + \alpha[\boldsymbol{p}(q) - {}_iIW^1(q-1)] \tag{5-9}$$

当输入矢量$\boldsymbol{P}$得到错误的分类时

$$a_k^2 = 1，\ t_k = 0，\ a_k^2 \neq t_k \tag{5-10}$$

IW^1的第i行可以按下式进行修正

$${}_iIW^1(q) = {}_iIW^1(q-1) - \alpha[\boldsymbol{p}(q) - {}_iIW^1(q-1)] \tag{5-11}$$

在调整IW^1的第i行时，其他行不变，所以输出误差反向传播到第1网络层，对IW^1的其他行没有影响。按上述方法进行修正的结果使得隐层的神经元趋近落入相应输入模式的输入矢量，远离其他模式的输入矢量，以构成其子分类。

所谓矢量量化，就是将矢量邻近的区域看做同一量化等级，用其中心值表示，即用少量的聚类中心表示原始数据。SOFM 和 LVQ 都具有矢量量化作用，不同的是 SOFM 的各中心（输出阵列中的神经元）的排列是有结构性的，即各相邻中心点对应的输入数据中的某种特征是相似的，而 LVQ 的中心没有这种排序功能。

注意

在 MATLAB 神经网络工具箱中，LVQ1 神经网络的第 1 网络层权值调整的学习函数是 learnlv1。

5.4.3 LVQ1 学习算法的改进

LVQ1 学习算法的改进（LVQ2）是在 LVQ1 的基础上进行的，它可以改善 LVQ1 学习结果的性能。

注意

在 MATLAB 神经网络工具箱中，LVQ2 神经网络的第 1 网络层权值调整的学习函数是 learnlv2。

LVQ2 的学习过程与 LVQ1 类似，在应用 LVQ1 进行学习后，再用 LVQ2 进行学习，不同的是，LVQ2 是针对最接近输入矢量的两个相邻神经元的权值进行的，其中一个神经元对应正确的分类模式，另一个神经元对应错误的分类模式，而输入向量位于定义的窗口时

$$\min\left(\frac{d_i}{d_j},\frac{d_j}{d_i}\right)>s,\quad s=\frac{1-w}{1+w} \tag{5-12}$$

式中，d_i、d_j 分别表示输入矢量 $\boldsymbol{p}$ 与 ${}_i\mathrm{IW}^1$、${}_j\mathrm{IW}^1$ 的欧几里得距离，w 在（0.2～0.3）之间取值。例如，当 w 取值为 0.25 时，s=0.6，那么，当 d_i 和 d_j 两个距离之比大于 0.6 时，则对 IW_i^1、IW_j^1 都需进行调整。

当第 i 个神经元对应的输出分类模式错误时，IW^1 的第 i 行可以按下式进行修正

$${}_i\mathrm{IW}^1(q)={}_i\mathrm{IW}^1(q-1)+\alpha[\boldsymbol{p}(q)-{}_i\mathrm{IW}^1(q-1)] \tag{5-13}$$

当第 j 个神经元对应的输出分类模式正确时，IW^1 的第 j 行可以按下式进行修正

$${}_j\mathrm{IW}^1(q)={}_j\mathrm{IW}^1(q-1)+\alpha[\boldsymbol{p}(q)-{}_j\mathrm{IW}^1(q-1)] \tag{5-14}$$

这样，如果给定两个很相近的输入矢量，其中一个对应正确的分类，而另一个对应错误的分类，则 LVQ2 也能对靠得非常近，甚至对刚刚可分的模式进行正确的分类，从而提高子分类结果的鲁棒性。

5.4.4　LVQ 神经网络的 MATLAB 程序

【例 5-5】画出具有 3×2 栅格拓扑结构的 SOFM，并以该神经网络完成对图 5-14 所示输入矢量模式的分类，分别画出当最大训练步长 epochs=40、60、100 时，调整权值后的神经元拓扑结构图。

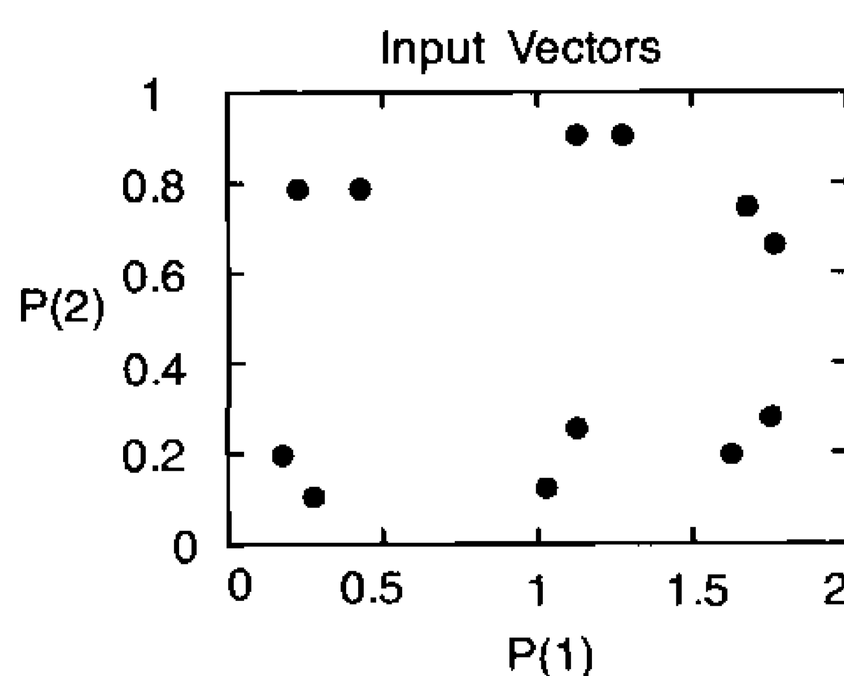


图 5-14　例 5-5 的输入矢量模式的分类

$$\boldsymbol{p}=\begin{bmatrix}0.1 & 0.3 & 1.2 & 1.1 & 1.8 & 1.7 & 0.1 & 0.3 & 1.2 & 1.1 & 1.8 & 1.7\\ 0.2 & 0.1 & 0.3 & 0.1 & 0.3 & 0.2 & 0.8 & 0.8 & 0.9 & 0.9 & 0.7 & 0.8\end{bmatrix}$$

（1）用 MATLAB 画出 3×2 栅格 SOFM 的特征映射图，如图 5-15 所示。

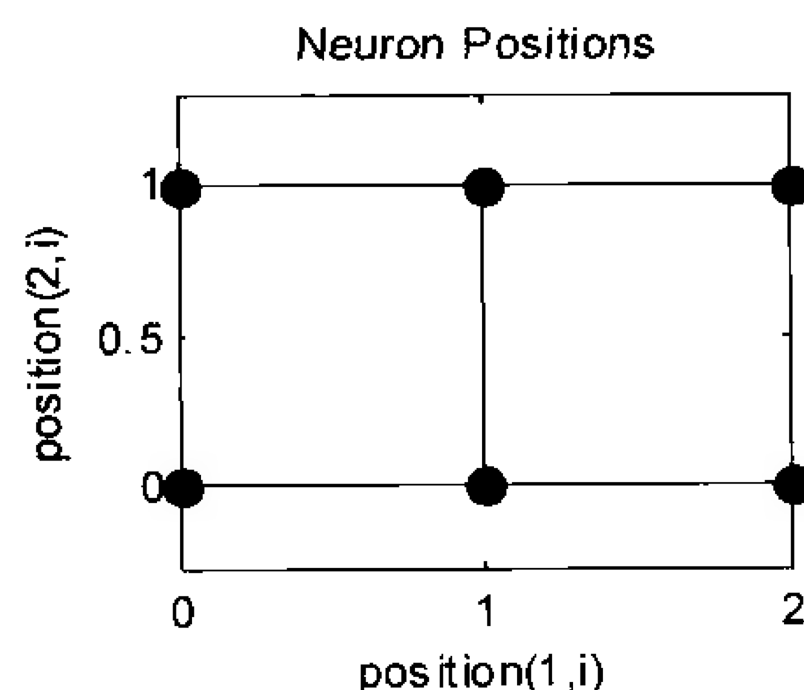


图 5-15　3×2 栅格 SOFM 特征映射图

绘制 3×2 栅格 SOFM 特征映射图的 MATLAB 程序如下。

```
pos=gridtop(3,2);
plotsom(pos)
```

（2）创建和训练 SOFM 神经网络的 MATLAB 程序设计如下。

```
clear all
%创建 SOFM 网络
net=newsom([0 2;0 1],[3 2],'gridtop');
%定义输入向量
P=[0.1 0.3 1.2 1.1 1.8 1.7 0.1 0.3 1.2 1.1 1.8 1.7
   0.2 0.1 0.3 0.1 0.3 0.2 0.8 0.8 0.9 0.9 0.7 0.8];
%绘制输入矢量
plot(P(1,:),P(2,:),'.g','markersize',18);
%训练 SOFM 网络
%设置训练步长的最大步长
net.trainParam.epochs=100;
net=train(net,P);
%绘制训练后的 SOFM 神经网络特征映射图
hold on;
plotsom(net.iw{1,1},net.layers{1}.distances);
hold off;
```

设置不同的最大步长进行训练，调整权值后的 SOFM 特征映射图如图 5-16 所示。

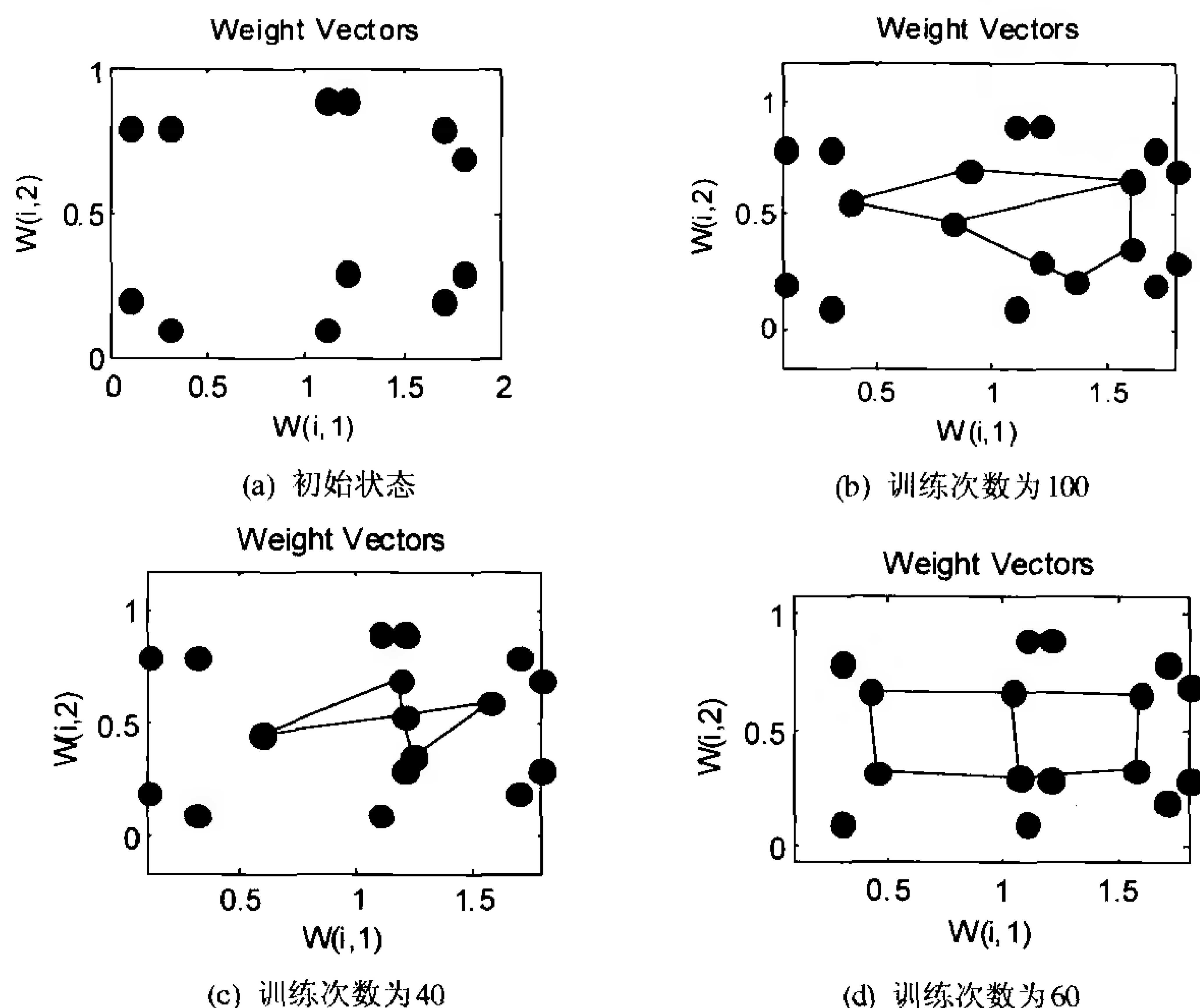


图 5-16　SOFM 神经网络权值的调整过程

SOFM 神经网络的学习，就是使权向量的方向朝着输入模式向量的方向进行调整，使各个权向量分别向各个聚类模式群的中心位置靠拢，同时，使网络权向量几何点的排列与竞争层各神经元的自然排列基本一致（拓扑结构一致）。从图 5-16 权值的调整过程看，调整的目标是使网络权向量几何点的排列与竞争层各神经元的拓扑结构基本一致，即为 3×2 栅格型结构。

（3）SOFM 神经网络的 MATLAB 仿真程序设计如下。

```
%定义输入向量
P=[0.1 0.3 1.2 1.1 1.8 1.7 0.1 0.3 1.2 1.1 1.8 1.7
   0.2 0.1 0.3 0.1 0.3 0.2 0.8 0.8 0.9 0.9 0.7 0.8];
%网络仿真
y=sim(net,P);
%输出仿真结果
yc=vec2ind(y)
```

仿真结果为

```
yc =
     4     4     5     5     6     6     1     1     2     2     3     3
```

因为输入向量模式具有明显的分类特征，所以 SOFM 网络很好地完成了分类。

【例 5-6】假设两种分类模式如图 5-17 所示，模式 1 表示竖线，模式 2 表示横线。试设计一个 LVQ 神经网络，完成这两种模式的分类。

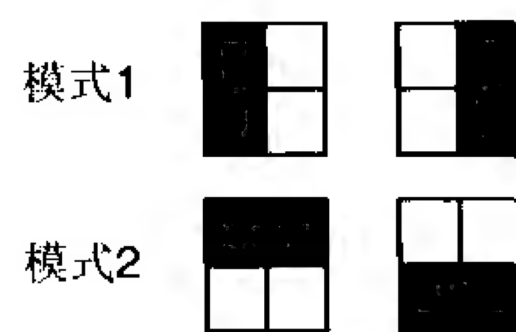


图 5-17　例 5-6 待分类的模式

（1）问题分析。

以图 5-18 表示输入向量的对应元素，则 LVQ 网络有 4 个输入元素，输出两类模式，所以线性层有 2 个神经元，设 01 表示横线，10 表示竖线。若以 0 表示线条划过的小方块，1 表示线条未划过的小方块，则图 5-17 所示的两类模式构成如下训练样本集。

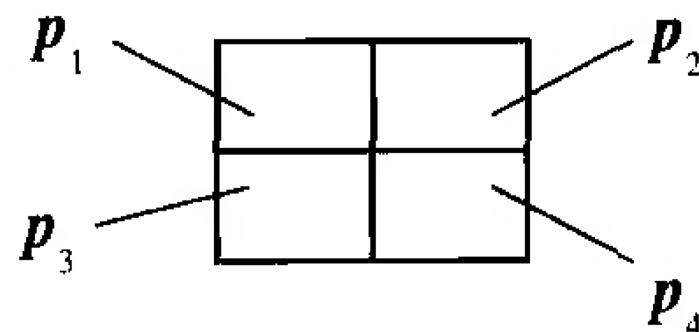


图 5-18　输入向量的对应元素

$$\boldsymbol{p}_1=\begin{bmatrix}0\\1\\0\\1\end{bmatrix},\boldsymbol{t}_1=\begin{bmatrix}1 & 0\end{bmatrix},\quad \boldsymbol{p}_2=\begin{bmatrix}1\\0\\1\\0\end{bmatrix},\boldsymbol{t}_2=\begin{bmatrix}1 & 0\end{bmatrix}$$

$$\boldsymbol{p}_3=\begin{bmatrix}0\\0\\1\\1\end{bmatrix},\boldsymbol{t}_3=\begin{bmatrix}0 & 1\end{bmatrix},\quad \boldsymbol{p}_4=\begin{bmatrix}1\\1\\0\\0\end{bmatrix},\boldsymbol{t}_4=\begin{bmatrix}0 & 1\end{bmatrix}$$

（2）创建和训练 LVQ 神经网络的 MATLAB 程序设计如下。

```
clear all
%定义输入向量和目标向量
P=[0 1 0 1;1 0 1 0;0 0 1 1;1 1 0 0]';
T=[1 1 0 0;0 0 1 1];
%创建 LVQ 网络
net=newlvq(minmax(P),4,[0.5 0.5],0.01,'learnlv1');
%训练 LVQ 网络
net=train(net,P,T);
TRAINR, Epoch 0/100
```

```
TRAINR, Epoch 4/100
TRAINR, Performance goal met.
```

运行结果为

```
TRAINR, Epoch 0/100
TRAINR, Epoch 4/100
TRAINR, Performance goal met.
```

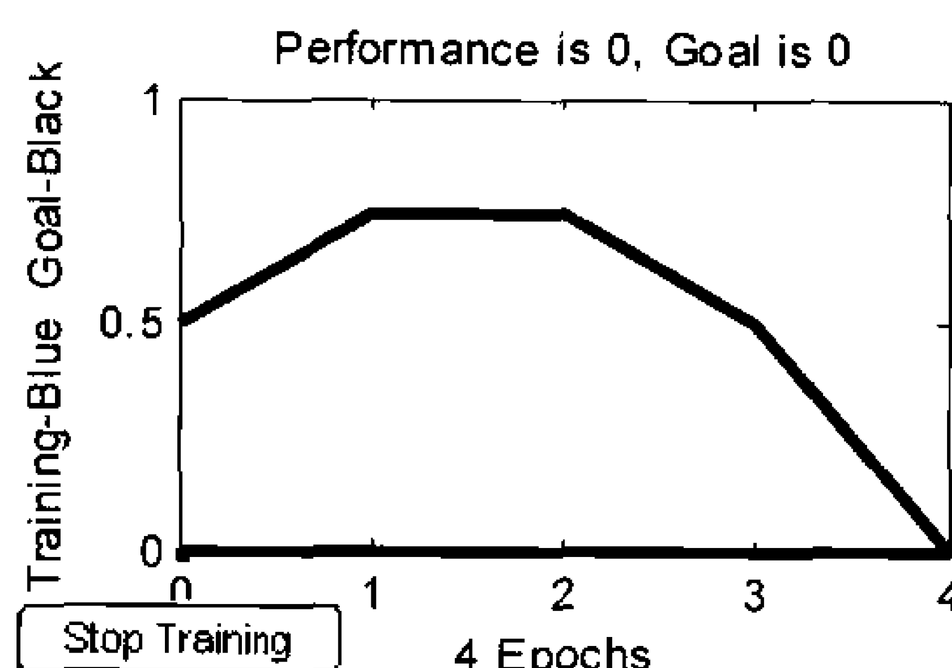


图 5-19　训练的误差性能曲线

（3）LVQ 神经网络的 MATLAB 仿真程序设计如下。

```
%定义输出向量
P=[0 1 0 1;1 0 1 0;0 0 1 1;1 1 0 0]';
%网络仿真
y=sim(net,P)
```

仿真结果为

```
y =

     1     1     0     0
     0     0     1     1
```

仿真结果表明很好地完成了分类。

5.5　对向传播网络

对向传播（Counter Propagation）网络，简称 CPN，是将 Kohonen 特征映射网络与 Grossberg 基本竞争型网络相结合，发挥各自特长的一种新型特征映射网络。这一网络是美国计算机专家 Robert Hecht-Nielsen 于 1987 年提出的。这种网络被广泛应用于模式分类、函数近似、统计分析和数据压缩等领域。

5.5.1　对向传播网络简介

CPN 网络结构如图 5-20 所示。由图可见，网络分为输入层、竞争层和输出层。输入层与竞争层构成 SOM 网络，竞争层与输出层构成基本竞争型网络。从整体上看，网络属于有教师型的网络，而由输入层和竞争层构成的 SOM 网络又是一种典型的无教师型神经网络。因此，这一网络既汲取了无教师型网络分类灵活、算法简练的优点，又采纳了有教师型网络的分类精细、准确的长处，使两种不同类型的网络有机地结合起来。

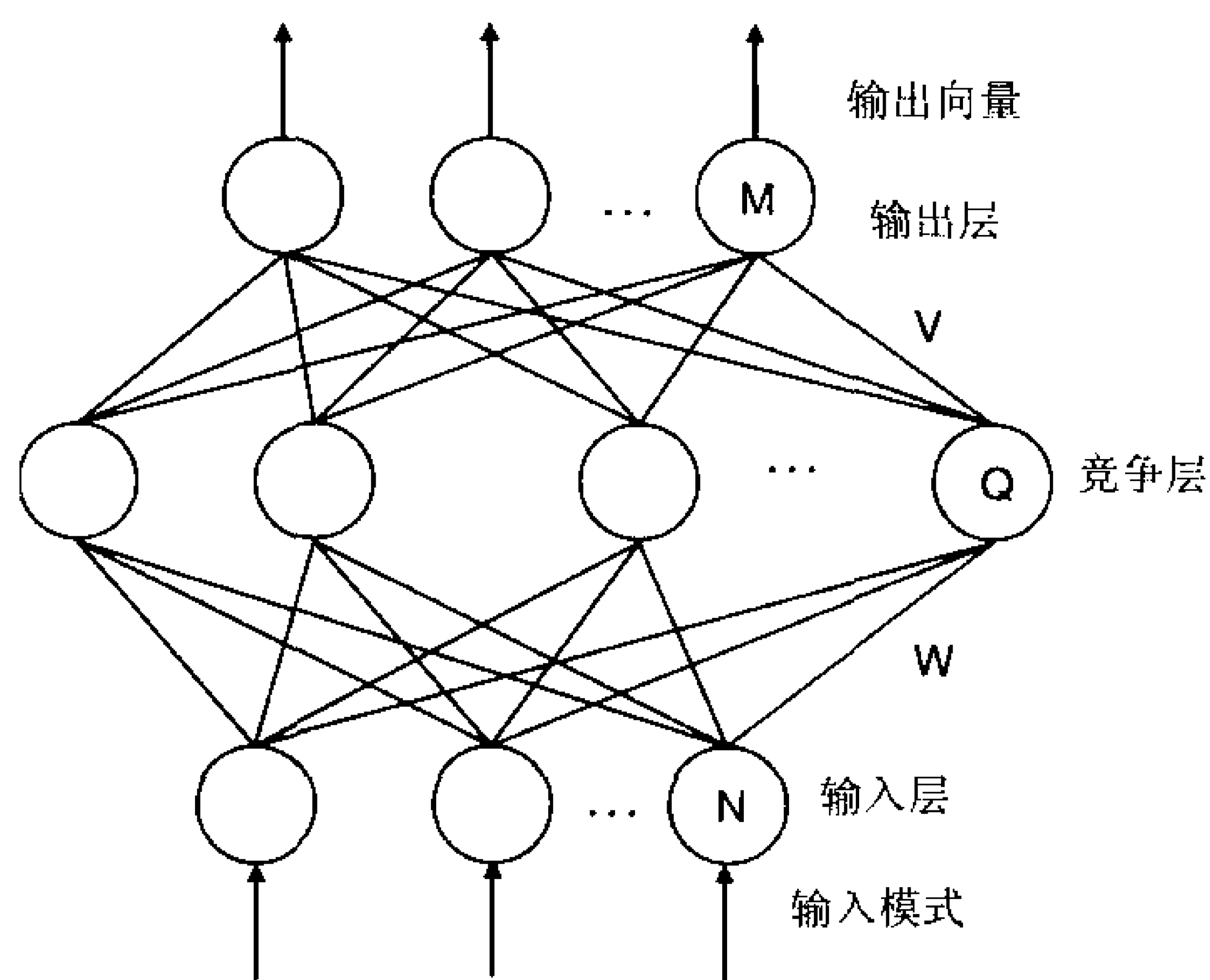


图 5-20　CPN 网络结构

CPN 的基本思想是，由输入层至输出层，网络按照 SOM 学习规则产生竞争层的获胜神经元，并按这一规则调整相应的输入层至竞争层的连接权；由竞争层到输出层，网络按照基本竞争型网络学习规则，得到各输出神经元的实际输出值，并按照有教师型的误差校正方法，修正由竞争层到输出层的连接权。经过这样的反复学习，可以将任意的输入模式映射为输出模式。

从这一基本思想可以发现，处于网络中间位置的竞争层获胜神经元及与其相关的连接权向量，既反映了输入模式的统计特性，又反映了输出模式的统计特性。因此，可以认为，输入、输出模式通过竞争层实现了相互映射，即网络具有双向记忆的功能。如果输入、输出采用相同的模式对网络进行训练，则由输入模式至竞争层的映射可以认为是对输入模式的压缩；而由竞争层至输出层的映射可以认为是对输入模式的复原。利用这一特性，可以有效地解决图像处理及通信中的数据压缩及复原问题，并可得到较高的压缩性。

接下来介绍 CPN 的学习及工作规则。假定输入层有 N 个神经元，p 个连续值的输入模式为 $\boldsymbol{A}_k=\left(a_1^k,a_2^k,\cdots,a_N^k\right)$，竞争层有 Q 个神经元，对应的二值输出向量为 $\boldsymbol{B}_k=\left(b_1^k,b_2^k,\cdots,b_Q^k\right)$，输出层有 $\boldsymbol{M}$ 个神经元，其连续值的输出向量为 $\boldsymbol{C}_k'=\left(c_1'^k,c_2'^k,\cdots,c_M'^k\right)$，目标输出向量为 $\boldsymbol{C}_k=\left(c_1^k,c_2^k,\cdots,c_M^k\right)$，以上 $k=1,2,\cdots,p$。由输入层至竞争层的连接权向量为 $\boldsymbol{W}_j=\left(w_{j1},w_{j2},\cdots,w_{jN}\right)$，$j=1,2,\cdots,Q$；由竞争层到输出层的连接权向量为 $\boldsymbol{V}_l=\left(v_{l1},v_{l2},\cdots,v_{lQ}\right)$，$l=1,2,\cdots,M$。网络学习和工作规则如下所述。

（1）初始化。将连接权向量 $\boldsymbol{W}_j$ 和 $\boldsymbol{V}_l$ 赋予区间[0,1]内的随机值。将所有的输入模式 $\boldsymbol{A}_k$ 进行归一化处理。

$$a_i^k=\frac{a_i^k}{\left\|\boldsymbol{A}_k\right\|},\quad \left\|\boldsymbol{A}_k\right\|=\sqrt{\sum_{i=1}^{N}\left(a_i^k\right)^2},\quad i=1,2,\cdots,N$$

（2）将第 k 个输入模式 $\boldsymbol{A}_k$ 提供给网络的输入层。

（3）将连接权向量 $\boldsymbol{W}_j$ 按照下式进行归一化处理。

$$w_{ji}=\frac{w_{ji}}{\left\|\boldsymbol{W}_j\right\|},\quad \left\|\boldsymbol{W}_j\right\|=\sqrt{\sum_{i=1}^{N}{w_{ji}}^2},\quad i=1,2,\cdots,N$$

（4）求竞争层中每个神经元的加权输入和。

$$s_j = \sum_{i=1}^{N} a_i^k w_{ji}\ ,\quad j = 1,2,\cdots,Q$$

（5）求连接权向量 $\boldsymbol{W}_j$ 中与 $\boldsymbol{A}_k$ 距离最近的向量 $\boldsymbol{W}_g$。

$$\boldsymbol{W}_g = \max_{j=1,2\cdots,Q} \sum_{i=1}^{N} a_j^k w_{ji} = \max_{j=1,2\cdots,Q} s_j$$

将神经元 g 的输出设定为 1，其余竞争层神经元的输出设定为 0。

$$b_j = \begin{cases} 1 & j = g \\ 0 & j \neq g \end{cases}$$

（6）将连接权向量 $\boldsymbol{W}_g$ 按照下式进行修正。

$$w_{gi}(t+1) = w_{gi}(t) + \alpha(a_i^k - w_{gi}(t)) \quad i = 1,2,\cdots,N$$

其中，$-1 < \alpha < 1$ 为学习率。

（7）将连接权向量 $\boldsymbol{W}_g$ 重新归一化，归一化算法同上。

（8）按照下式修正竞争层到输出层的连接权向量 $\boldsymbol{V}_l$。

$$v_{li}(t+1) = v_{li}(t) + \beta b_j(c_l - c_l') \quad l = 1,2,\cdots,M\ ,\quad j = 1,2,\cdots,Q$$

其中，$-1 < \beta < 1$ 为学习率。由步骤（5）可将上式简化为

$$v_{lg}(t+1) = v_{lg}(t) + \beta b_j(c_l - c_l')$$

由此可见，只需要调整竞争层获胜神经元 g 到输出层神经元的连接权向量 $\boldsymbol{V}_g$ 即可，其他连接权向量保持不变。

（9）求输出层各神经元的加权输入，并将其作为输出神经元的实际输出值，$c_l' = \sum_{j=1}^{Q} b_j v_{lg}$，$l = 1,2,\cdots,M$，同理可将其简化为 $c_l' = v_{lg}$。

（10）返回步骤（2），直到将 p 个输入模式全部提供给网络。

（11）令 $t = t+1$，将输入模式 $\boldsymbol{A}_k$ 重新提供给网络学习，直到 $t = T$。其中 T 为预先设定的学习总次数，一般取 500<T<100000。

5.5.2 对向传播网络的 MATLAB 程序

【例 5-7】

这里举一个非常简单而且与日常生活相关的例子来说明 CPN 网络的应用。现在需要创建一个 CPN 网络，其任务是在已知一个人本星期应该完成的工作量和此人当时的思想情绪状态的情况下，对此人星期日下午的活动安排提出建议。

按照一般情况，将工作量分为 3 个档次，即“没有”、“有一些”和“很多”，所对应的量化值分别为 0.0、0.5 和 1.0；把思想情绪也分为 3 个水平，即“低”、“一般”和“高”，所对应的量化值分别为 0.0、0.5 和 1.0。可选择的活动有 5 个，即在家里看画报、去商场购物、到公园散步、与朋友一起吃饭和干工作。工作量和思想情绪状态一共有 6 种组合，这 6 种组合分别对应各自的最佳活动选择。样本模式如表 5-3 所示。

把这组训练样本提供给网络进行充分学习后，网络就具有了一种“内插”功能，即当网络输入一对在 [0,1] 区间中反映工作量和情绪的量化值后，网络将自动根据原有的记忆，找出对应于这对量化值的最佳活动选择，以输出模式的形式提供给用户作为决策参数。

表 5-3　网络训练样本模式

工 作 量	思想情绪	活动安排	目标输出
没有 0.0	低 0.0	看画报	10000
有一些 0.5	低 0.0	看画报	10000
没有 0.0	一般 0.5	购物	01000
很多 1.0	高 1.0	公园散步	00100
有一些 0.5	高 1.0	吃饭	00010
很多 1.0	一般 0.5	工作	00001

实际上，不光 CPN 网络具有这种“内插”功能，BP 网络、SOM 网络都具有这种功能。从模式识别的角度上讲，这些网络具有对输入模式进行分类的功能。

可惜的是，对功能如此强大的 CPN 网络，神经网络工具箱中竟然没有与之支持的函数工具。但是，既然 5.5.1 节中已经给出了有关 CPN 的学习和训练算法过程，因此，我们可以利用 MATLAB 强大的数学计算功能，实现解决该问题的 CPN 网络。

根据题意，该网络的输入层应该有 2 个神经元，输出层应该有 5 个神经元。为了更加准确地解决问题，将竞争层神经元设置为 18 个。网络结构如图 5-21 所示。

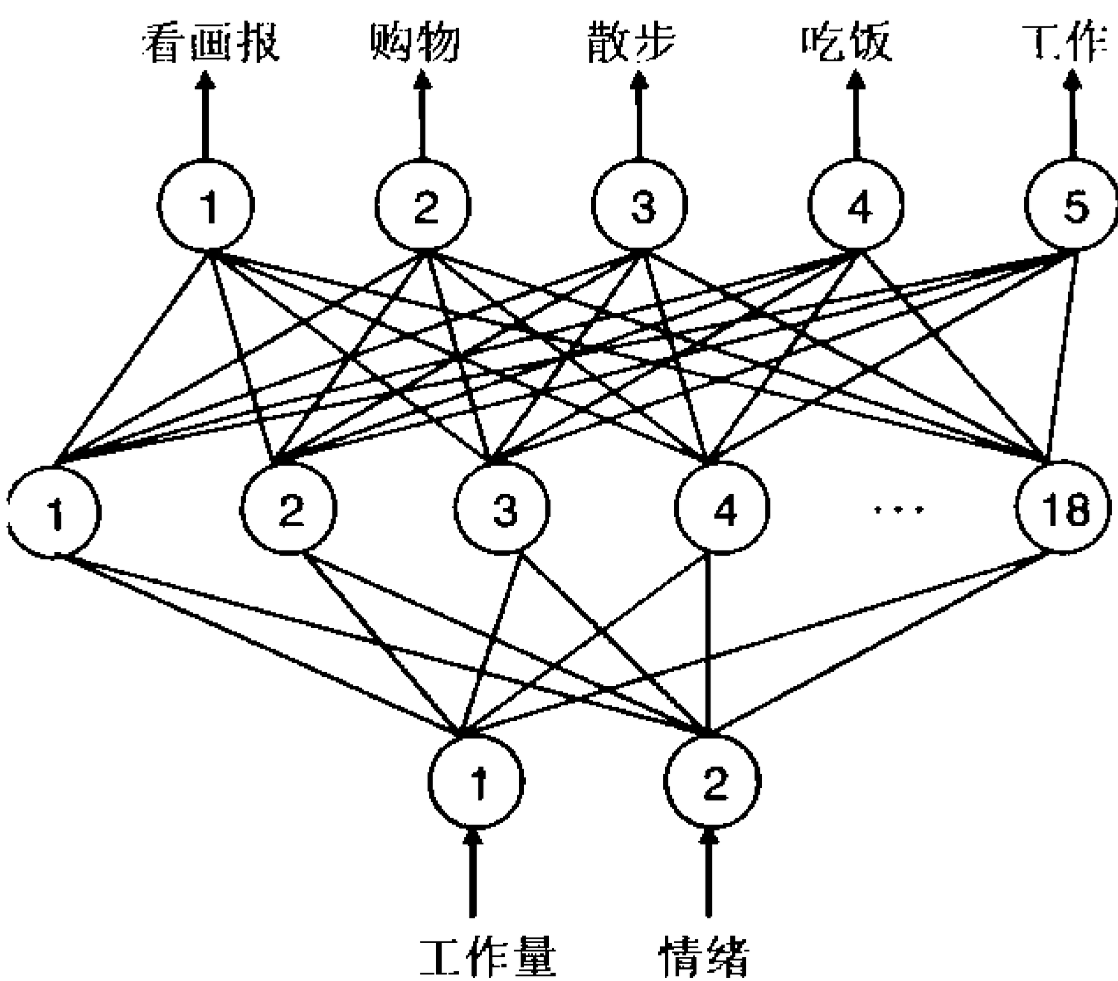


图 5-21　星期日下午活动安排决策 CPN 网络

由表 5-3 可得，网络的输入向量为

P=[0 0;0.5 0.5;0 0.5;1 1;0.5 1;1 0.5];

目标向量为

T=[1 0 0 0 0;1 0 0 0 0;0 1 0 0 0;0 0 1 0 0;0 0 0 1 0;0 0 0 0 1]';

下面对网络进行一个周期的学习。令输入层和竞争层之间的连接权向量矩阵用 **W** 表示，竞争层和输出层之间的权向量矩阵用 **V** 表示。可知 **W** 为一个 18×2 的矩阵，**V** 是一个 5×18 的矩阵。学习速率设定为 0.1。

（1）初始化。利用 MATLAB 中的随机数产生函数 **W** 和 **V**，并赋以区间[0,1]之间的随机值，代码为

```
W=rands(18,2)/2+0.5;
V=rands(5,18)/2+0.5;
```

由于函数 rands 产生的随机数位于区间［-1,1］之间，所以这里做了这样的处理，使得产生的随机数既不影响随机性能，又位于区间[0，1]中。

对输入向量进行归一化处理

```
W=rands(18,2)/2+0.5;
V=rands(5,18)/2+0.5;
for i=1:6
    if(P(i,:)==[0 0])
        P(i,:)=P(i,:)
    else
        P(i,:)=P(i,:)/norm(P(i,:));
    end
end
```

之所以要在循环中进行数据判断，是因为向量[0 0]是无法归一化处理的，比如 **P** 的第一组元素，就是正在用的这一组。

（2）将第一个输入样本［0 0］提供给网络的输入层神经元。

（3）对连接权向量 **W** 进行归一化处理。

（4）求每一个竞争层神经元的加权输入 s_j， $j=1,2,\cdots,18$。

```
for i=1:18
    W(i,:)=W(i,:)/norm(W(i,:));
    s(i)=P(1,:)*W(i,:)';
end
```

循环语句中第一句用于对连接权向量 **W** 进行归一化处理，第二句可以求出竞争层每个神经元的输出。结果为

```
s =
    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0
```

（5）求连接权向量 **W** 中与［0 0］距离最近的向量，由于输出全部为 0，所以可任选一个权值向量 $\boldsymbol{W}_g$，这里选为 18，并将该神经元的输出设定为 1。

```
for i=1:18
    W(i,:)=W(i,:)/norm(W(i,:));
    s(i)=P(1,:)*W(i,:)';
end
temp=max(s);
for i=1:18
    if temp==s(i)
```

```
            count=i;
        end
    end
    %将所有竞争层神经元的输出置为 0
    for i=1:18
        s(i)=0;
    end
    %选中的神经元输出为 1
    s(count)=1;
```

（6）调整连接权向量 $\boldsymbol{W}_{18}$，并重新将其归一化。

```
W(count,:)=W(count,:)+0.1*[P(1,:)-W(count,:)];
W(count,:)=W(count,:)/norm(W(count,:))
```

输出结果为

```
W(18,:)=
        0.9997      0.0251
```

经检验，此时的 $\boldsymbol{W}_{18}$ 确实已经归一化了。

（7）调整竞争层神经元到输出层神经元之间的连接权向量 $\boldsymbol{V}$。

```
V(:,count)=V(:,count)+0.1*(T(1,:)'-T_out(1,:)');
```

由于此时输出层神经元的输出域目标向量是一致的，所以这里的权值是没有经过调整的，等到来了下一个模式后，权值才会真正得到调整。

（8）计算输出层各神经元的加权输入，并将其作为神经元的实际输出值。

```
T_out(1,:)=V(:,count)';
```

第一组实际输出就等于竞争层中第 18 个神经元与输出层各神经元之间调整后的连接权值。

（9）返回步骤（2），将输入向量中的［0.5 0.5］提供给网络。

（10）继续学习，直到训练次数达到设定的最大值。

CPN 网络训练结束后，按照以下步骤进行网络回想。

（1）将输入模式 $\boldsymbol{A}$ 提供给网络的输入层。

（2）根据下式求出竞争层的获胜神经元 g。

$$b_g = \max_{i=1,2,\cdots,Q}\left(\sum_{i=1}^{N} w_{ji}a_i\right)$$

（3）令 $b_g = 1$，其余的输出都等于 0。按照下式求得输出层各神经元的输出。

$$c_j = v_{jg}b_g$$

由此产生了输出模式 $\boldsymbol{C} = (c_1, c_2, \cdots, c_M)$，从而就得到了输入模式 $\boldsymbol{A}$ 的分类结果。

```
clear all
%初始化正向权值 W 和反向权值 V
W=rands(18,2)/2+0.5;
V=rands(5,18)/2+0.5;
%输入向量 P 和目标向量 T
```

```
P=[0 0;0.5 0.5;0 0.5;1 3;0.5 1;1 0.5];
T=[1 0 0 0 0;1 0 0 0 0;0 1 0 0 0;0 0 1 0 0;0 0 0 1 0;0 0 0 0 1]';
T_out=T;
%设定学习步数为 1000 次
epoch=1000;
%归一化输入向量 P
for i=1:6
    if P(i,:)==[0 0]
        P(i,:)=P(i,:);
    else
        P(i,:)=P(i,:)/norm(P(i,:));
    end
end
%开始训练
while epoch>0
    for j=1:6
        %归一化正向权值 W
        for i=1:18
            W(i,:)=W(i,:)/norm(W(i,:));
            s(i)=P(j,:)*W(i,:)';
        end
        %求输出最大的神经元，即获胜神经元
temp=max(s);
for i=1:18
    if temp==s(i)
        count=i;
    end
end
%将所有竞争层神经元的输出置为 0
for i=1:18
    s(i)=0;
end
%选中的神经元输出为 1
s(count)=1;
%权值调整
W(count,:)=W(count,:)+0.1*[P(j,:)-W(count,:)];
W(count,:)=W(count,:)/norm(W(count,:));
V(:,count)=V(:,count)+0.1*(T(j,:)'-T_out(j,:)');
%计算网络输出
T_out(j,:)=V(:,count)';
    %end
    %训练次数递减
    epoch=epoch - 1;
end
%训练结束
T_out
```

```
%网络回想
%网络的输入模式 Pc
Pc=[0.5 1;1 3];
%初始化 Pc
for i=1:2
    if Pc(i,:)==[0 0]
        Pc(i,:)=Pc(i,:);
    else
        Pc(i,:)=P(i,:)/norm(Pc(i,:));
    end
end
%网络输出
Outc=[0 0 0 0 0;0 0 0 0 0];
for j=1:2
    for i=1:18
        sc(i)=Pc(j,:)*W(i,:)';
    end
    tempc=max(sc);
    for i=1:18
        if tempc==sc(i)
            countp=i;
        end
        sc(i)=0;
    end
    sc(countp)=1;
    Outc(j,:)=V(:,countp)';
end
%回想结束
Outc
```

输出结果为

```
T_out =
    1    1    0    0    0    0
    0    0    1    0    0    0
    0    0    0    1    0    0
    0    0    0    0    1    0
    0    0    0    0    0    1
```

由此可见，经过 1000 次训练后，网络的实际输出和目标输出就一致了，这说明训练过程是有效的。

```
Outc =
    0.3050    0.8744    0.0150    0.7680    0.9708
    0.3784    0.8600    0.8537    0.5936    0.4966
```

Outc 是网络回想的输出，实际上也就是网络测试的结果，在这里给出了两种特定的组合状态，即［0.5 1］和［1 1］的组合，这两种组合分别对应吃饭和到公园散步，可见，网络给出了正确的建议。

注意

在以上代码中，训练输入向量 ***p*** 和回想输入向量 Pc 中的 [1 1] 都被 [1 3] 所替代，这是因为经过归一化处理后，***p*** 中两个本来不一致的样本 [0.5 0.5] 和 [1 1] 变得一致了，而它们对应的输出向量却并不一致。因此，将其用 [1 3] 就是为了避免这种情况，对结果并没有影响。

第 6 章　反馈神经网络及其应用

反馈型神经网络（recurrent networks）是一种从输出到输入具有反馈连接的神经网络，其结果比前馈网络要复杂得多。典型的反馈神经网络有：Elman 网络和 Hopfield 网络。反馈网络的输入包含有延迟的输入或者输出数据的反馈。由于有反馈的输入，所以它是一种反馈动力学系统。这种系统的学习过程就是它的神经元状态的变化过程，这个过程最终会达到一个神经元状态不变的稳定态，也标志着学习过程结束。反馈网络的这种动态学习特性，主要由网络的反馈形式决定。反馈网络的反馈形式是比较多样化的，有输入延迟的、单层输出的、神经元自反馈的、两层之间互相反馈的等。

6.1　反馈神经网络的基本概念

反馈网络包括输入延迟和输出反馈两种类型，其他的网络称为静态网络。为了对它们的不同之处有个清晰的理解，下面用一个例子来详细说明。

【例 6-1】静态网络和反馈网络对同一个输入序列的不同响应。

```
%输入的序列
p={0 0 0 1 1 1 0 0 0 0 0};
%画出序列图
stem(cell2mat(p));
```

序列图如图 6-1 所示。接下来，创建一个静态网络，并且得出这个静态网络对该序列的响应。用下面的命令创建一个单层，单神经元，没有偏置值，权重为 2 的线性网络。

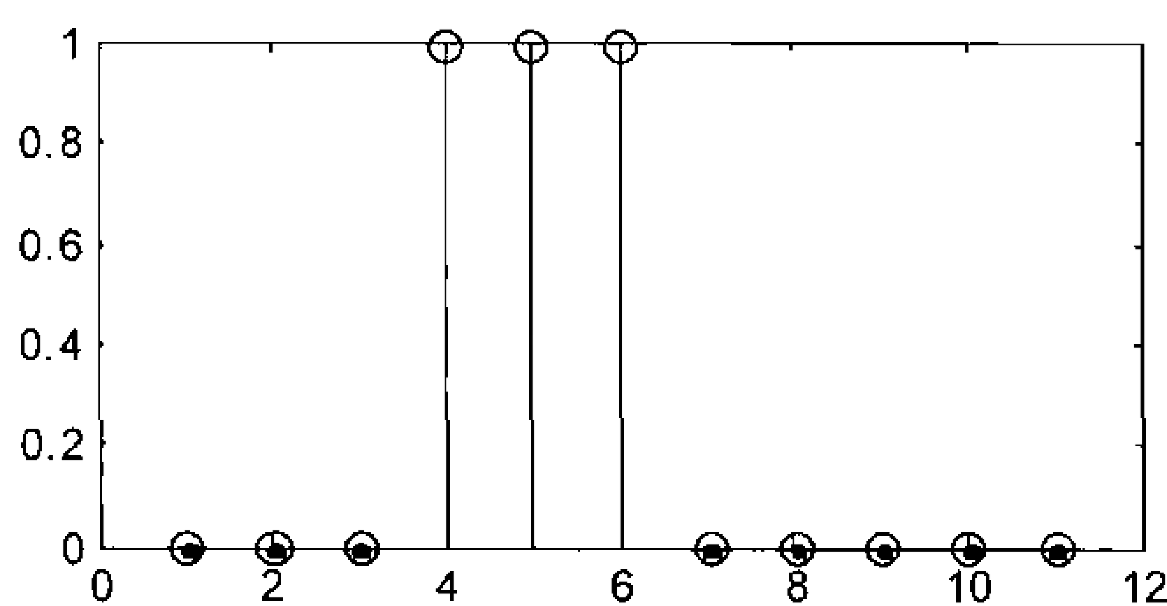


图 6-1　输入序列图

```
clf
%创建一个单层的线性网络
net=newlin([-1 1],1);
%偏置值为零
net.biasConnect=0;
%权值为 2
net.iw{1,1}=2;
```

用上面创建的线性网络仿真输入序列，并画出序列图，如图 6-2 所示。

```
%网络仿真
A=sim(net,p);
```

```
%画出序列图
stem(cell2mat(A));
```

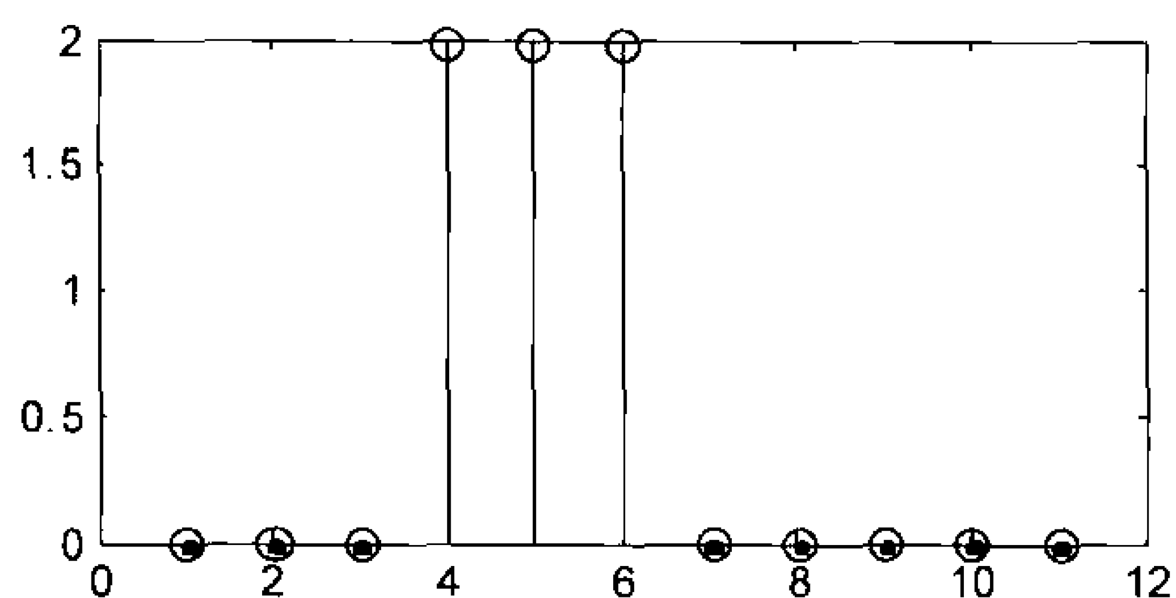


图 6-2　静态网络输出序列图

图 6-2 是静态线性网络对输入序列的仿真结果。从图中可以看出，静态网络的响应所持续的时间和输入序列一样，即静态网络在任意时间点的响应只依赖于同一时间点的输入。

接下来，创建一个反馈网络。这个反馈网络相当于一个输入有延迟的单层、单神经元的线性网络。

```
%创建输入有延迟的反馈网络
net=newlin([-1 1],1,[0 1]);
%偏置值为零
net.biasConnect=0;
%网络权值为[1 1]
net.iw{1,1}=[1 1];
```

再一次用该网络仿真输入序列。

```
A=sim(net,p);
stem(cell2mat(A));
```

从图 6-3 可以看出，反馈网络的响应在时间上延迟得比输入序列要长。因为反馈网络在任意时刻的响应不仅依赖于当前的输入，也依赖于以前的输入，所以它是有记忆效应的。

现在考虑一个输入中包括输出反馈的网络，即层反馈的网络。可以用第 3 章讨论过的 newnarx 命令创建这种网络。

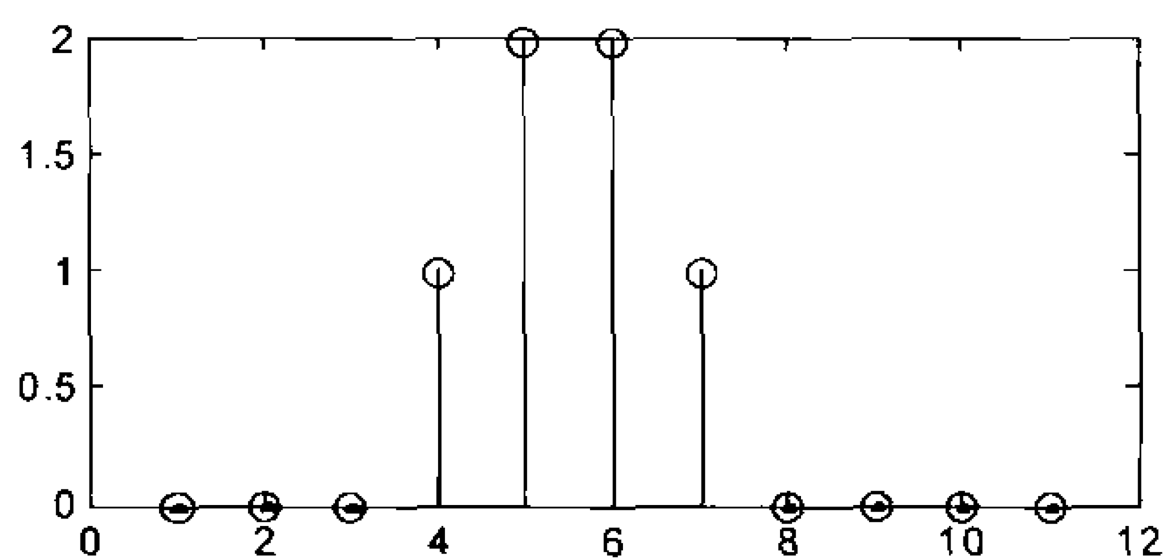


图 6-3　输入延迟反馈网络输出序列图

```
%创建单层、单神经元、输出层延迟为 1 的层反馈网络
net=newnarx([-1 1],0,1,1 {'purelin'});
%偏置为 0，输出层权重为 0.5，输入层权重为 1
net.biasConnect=0;
net.lw{1}=0.5;
net.iw{1}=1;
%对输入序列仿真
A=sim(net,p);
stem(cell2mat(A))
```

从图 6-4 可以注意到，层回馈的反馈网络比输入延迟的反馈网络有一个更长的输出响应。这是因为在时间上输出总是要晚于输入，所以输出有反馈的层反馈网络的输出响应延迟的时间比输入有反馈的网络延迟的时间要更长。

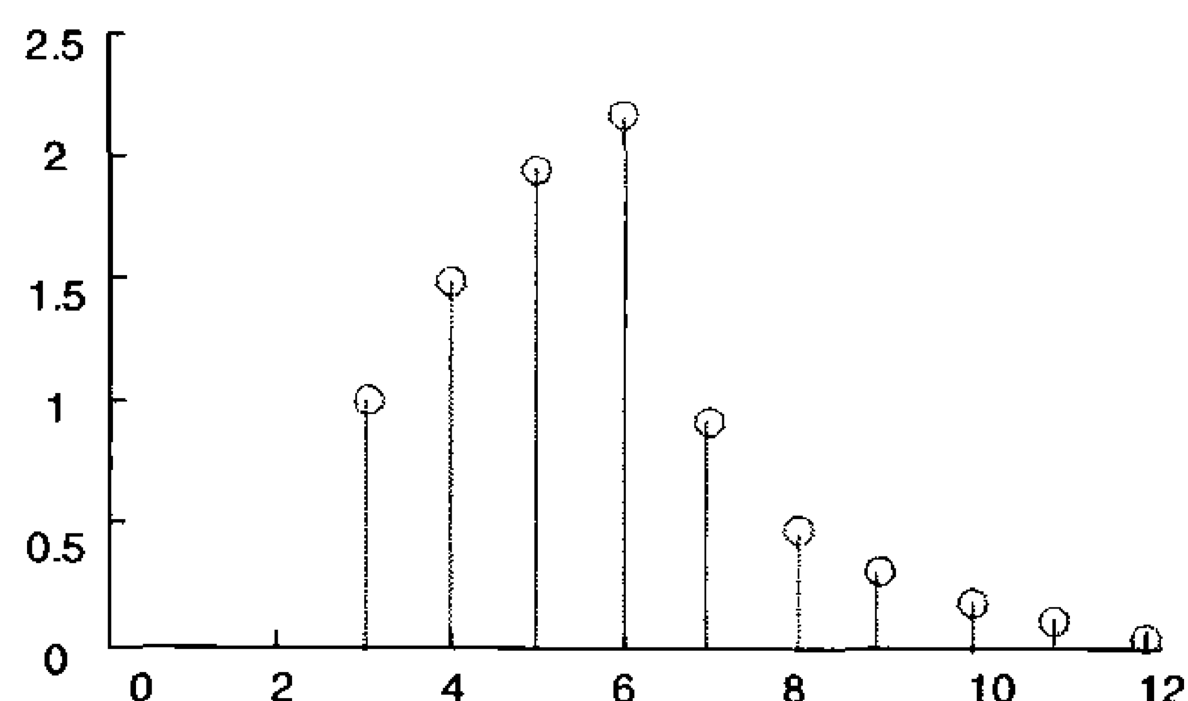


图 6-4　层反馈的反馈网络输入序列图

通过这一节，可以对反馈网络的基本特性有一个初步的了解，下面就具体的例子来讲解。

6.2　Elman 神经网络及 MATLAB 程序

Elman 网络是两层反向传播网络。隐层和输入向量连接的神经元，其输出不仅作为输出层的输入，而且还连接隐层内的另外一些神经元，反馈至隐层的输入。由于其输入表示了信号的空域信息，而反馈支路是一个延迟单元，反映了信号的时域信息，所以 Elman 网络可以在时域和空域进行模式识别。

6.2.1　Elman 神经网络

Elman 网络由若干个隐层和输出层构成，并且隐层存在反馈环节，其结构如图 6-5 所示。图中，模块 D 表示时延环节，隐层神经元采用 tansig 函数作为传递函数，输出层传递函数为纯线性函数 purelin，这两层神经元的传递函数可以在建立网络时由用户自己指定。当隐层神经元足够多时，Elman 网络的这种结构可以保证网络以任意精度逼近任意的非线性函数。

值得注意的是，Elman 网络不同于通常的两层网络，其第一网络层有一个反馈节点，该节点的延迟量存储了前一时刻的值，从而应用于当前时刻的计算。所以即使是具有相同权值和阈值的 Elman 网络，如果其反馈状态不同，那么对于同样的输入向量，其同一时刻的输出也可能不同。

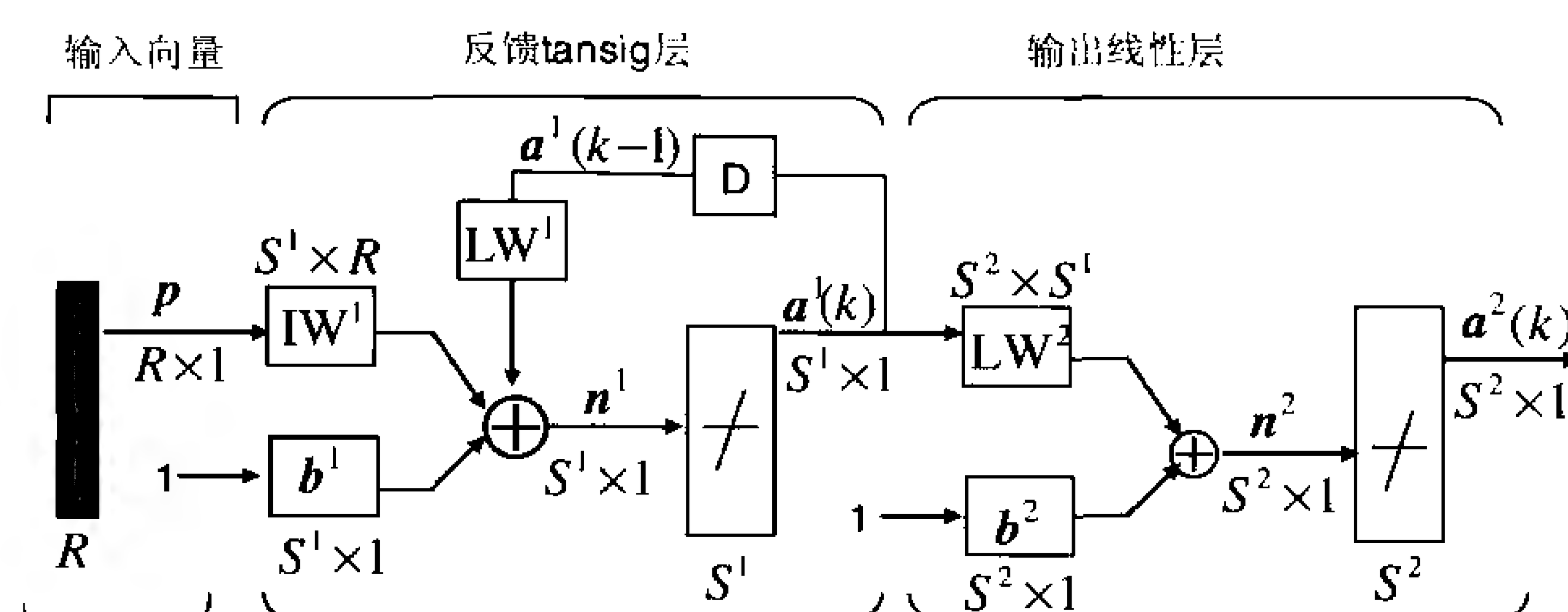


图 6-5　Elman 神经网络模型

因为 Elman 网络能够存储供将来时刻使用的信息，所以它既可以学习时域模式，也可以学习空域模式；它既可以训练后对模式产生响应（模式的空间分类结果），也可以产生模式输出（模式的时域变化关系）。

以 newelm 定义的 Elman 网络为例，其反向传播训练算法的默认函数为 trainbfg，还可以用 trainlm，但其处理速度太快，这在 Elman 网络中是不必要的，效果也不一定好。默认的权值和阈值的反向传播函数为 learngdm，默认的误差性能函数为 mse。

网络建立后，每个网络层的权值和阈值都以 Nguyen-Widrow 网络层初始化方法进行初始化，实现函数为 initnw。

Elman 网络的训练可采用 train 或 adapt 两个函数中的任意一个。当采用函数 train 时，每一步迭代过程按以下步骤进行。

（1）在网络输入端先输入所有的输入序列，然后计算输出结果，并与目标序列进行比较，从而产生一误差序列。

（2）在每一次迭代中，误差被反向传播，以确定每一个权值和阈值的误差梯度，实际上梯度的计算是近似的，因为经由延迟反馈支路对权值和阈值误差的贡献是忽略了的。

（3）该梯度用于对用户选择的反向传播训练函数进行权值修正。建议采用训练函数 traingdx。

当采用函数 adapt 时，迭代过程与采用函数 train 时基本一样，只是权值的修正函数建议采用学习函数 learngdm。

Elman 网络没有其他网络的可靠性高，因为训练或调整都是以误差梯度的近似值进行的。

6.2.2 Elman 神经网络的 MATLAB 程序

【例 6-2】Elman 神经网络用于峰值检波。

（1）问题分析。

振幅调制（AM）是通信系统一种最常见的模拟通信方式，在接收端往往采用峰值检波，这里以 Elman 神经网络来实现。训练样本集中的输入样本采用三角波调制的调幅波形，调幅度为 100%，载波频率约为 3.18Hz(20rad/s)，调制信号频率为 0.11Hz(0.67rad/s)；目标向量为调制信号，即调幅波的包络，如图 6-6 所示。

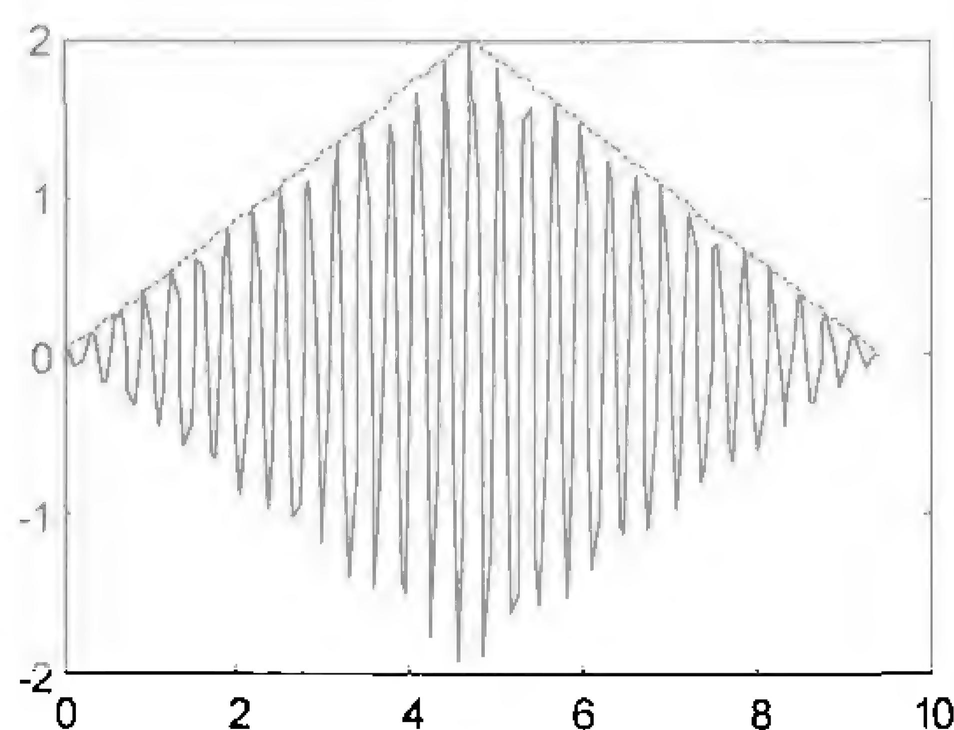


图 6-6　Elman 网络的输入样本和目标向量波形

采用调幅度为 100%、调制信号为三角波调制的调幅波形，可以使已调波信号从 0 变到最大值，从而使训练后的网络能够较完整地反映不同调幅度、不同波形的情况，使网络解调的性能更好。

载波频率选为 3.18Hz(20rad/s)，调制信号频率选为 0.11Hz(0.67rad/s)，这完全是为了使绘制的波形便于读者观察，同时减小网络输入向量的规模而设置的，实际上载频要高得多，调制信号也有一定的频带宽度。

从 Elman 神经网络的机理上看，可以将 AM 已调波信号的输入看成是时域中的信号，网络在时域中先对其进行识别；而已调波的包络可以看成二维平面上的曲线，即为空域中的信号模式，Elman 神经网络在空域中对输入向量的模式分类成峰值检波的输出。

Elman 神经网络的 MATLAB 程序代码如下。

```
clear all;
%定义输入向量和目标向量
Time1=0:0.05:1.5*pi;
T1=Time1/(1.5*pi)-0.5;
Time2=1.5*pi:0.05:3*pi;
T2=1.5-Time2/(1.5*pi);
Time=[Time1 Time2];
%三角波（目标向量）
T=2*[T1 T2];
%调幅波（输入向量）
p=(1+T).*cos(20*Time);
%创建 Elman 网络
net=newelm(minmax(p),[20 1],{'tansig','purelin'},'traingdx');
%训练 Elman 网络
Pq=con2seq(p);        %将输入向量矩阵转换为输入序列
Tq=con2seq(T);        %将目标向量矩阵转换为目标序列
plot(Time,p,Time,1+T,'r--');
pause;
net.trainParam.epochs=500;
```

训练的误差性能曲线如图 6-7 所示，到达最大训练步长 500 时，其均方误差 mse=0.0186686。

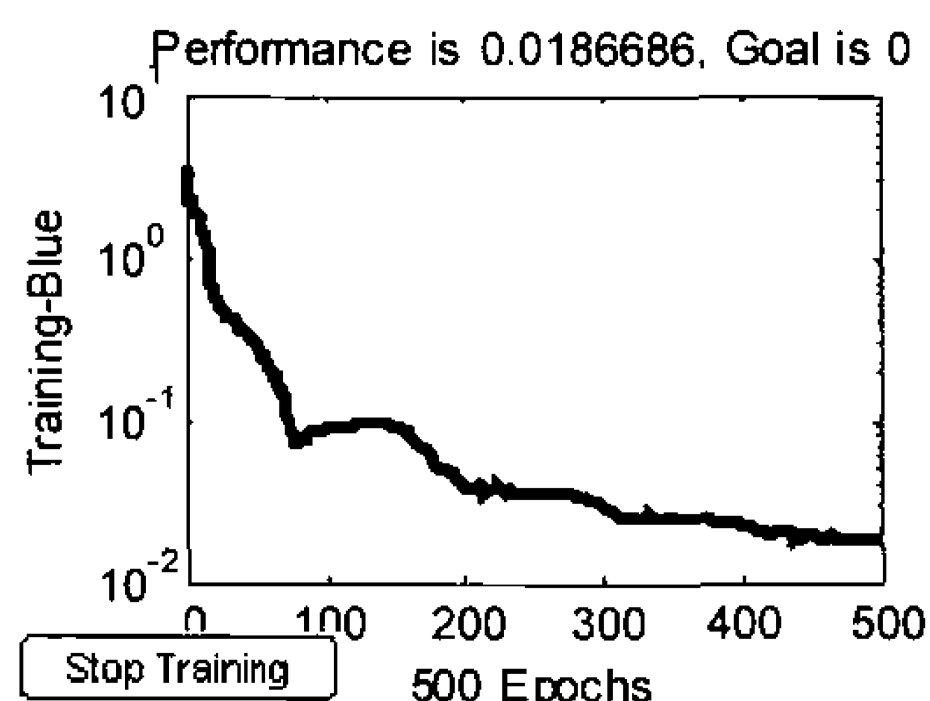


图 6-7　Elman 网络的训练误差性能曲线

运行结果如下。

```
TRAINGDX, Epoch 0/500, MSE 0.456495/0, Gradient 3.78279/1e-006
TRAINGDX, Epoch 25/500, MSE 0.231006/0, Gradient 0.573157/1e-006
……
TRAINGDX, Epoch 500/500, MSE 0.0186686/0, Gradient 0.0196215/1e-006
TRAINGDX, Maximum epoch reached, performance goal was not met.
```

（2）Elman 神经网络的 MATLAB 仿真程序设计。

我们选用了三角波、正弦波和矩形波 3 种调制信号形成的已调波形作为测试信号，对所设计的 Elman 神经网络进行仿真，其仿真程序如下。

```
%以三角波调制进行仿真
Time1=0:0.05:2*pi;
T1=Time1/(2*pi)-0.5;
Time2=2*pi:0.05:4*pi;
T2=1.5-Time2/(2*pi);
```

```
Time=[Time1 Time2];
```

仿真结果显示如图 6-8 所示。

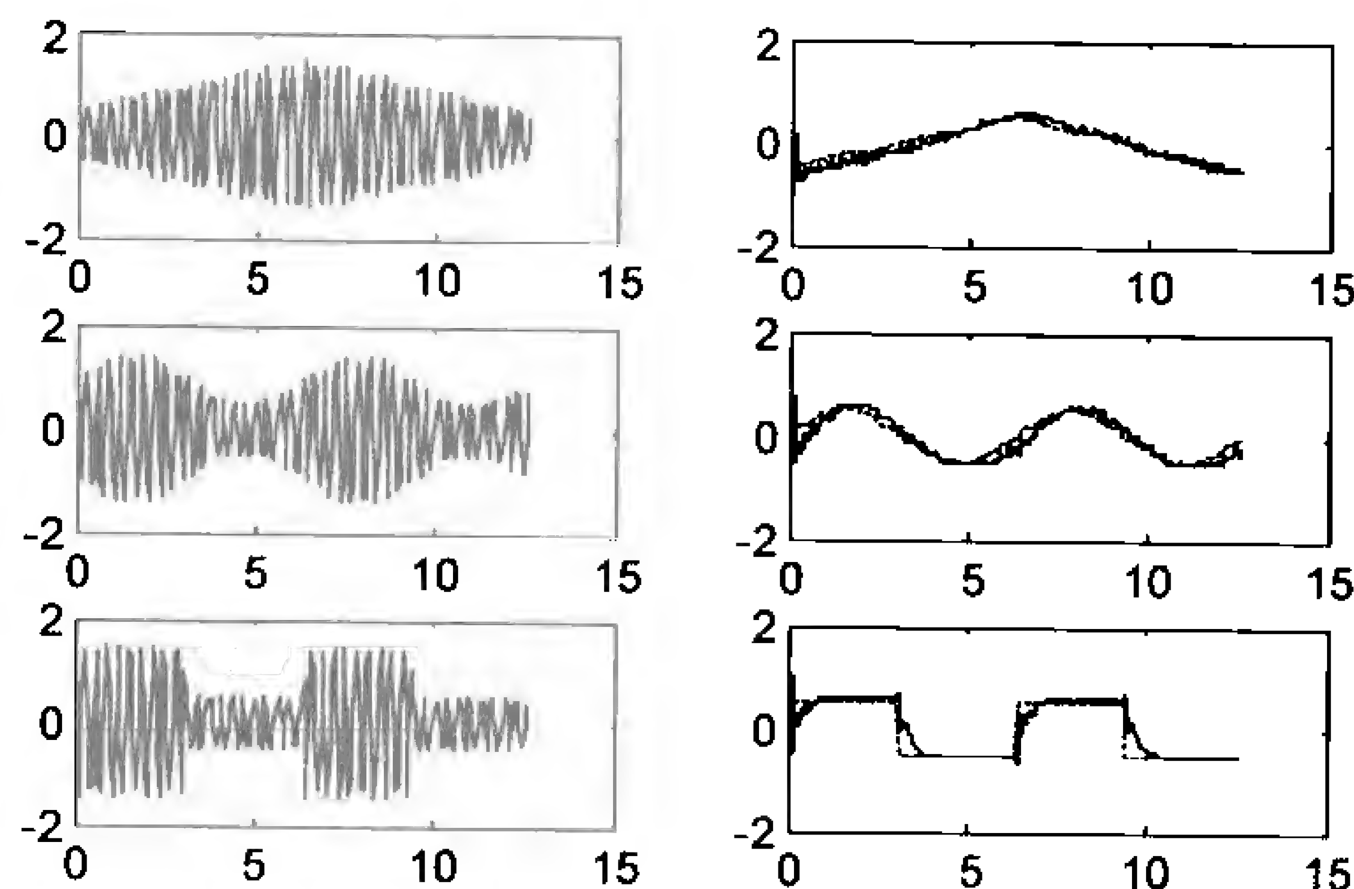


图 6-8　Elman 神经网络对于不同调制信号的仿真结果显示

```
%三角波（目标向量）
T=[T1 T2];
%调幅波（输入向量）
p=(1+T).*cos(20*Time);  %形成三角波调制的已调波信号
subplot(321);
plot(Time,p);
Pq=con2seq(p);
A=sim(net,Pq);
Y=seq2con(A);
subplot(322);
plot(Time,Y{1},Time,T,'r--');
%以正弦波调制进行仿真
T=0.5*sin(Time);
p=(1+T).*cos(20*Time);
subplot(323);
plot(Time,p);
Pq=con2seq(p);
A=sim(net,Pq);
Y=seq2con(A);
subplot(324);
plot(Time,Y{1},Time,T,'r--');
%以矩形波调制进行仿真
T=0.5*sign(sin(Time));
p=(1+T).*cos(20*Time);
subplot(325);
plot(Time,p);
Pq=con2seq(p);
A=sim(net,Pq);
Y=seq2con(A);
subplot(326);
plot(Time,Y{1},Time,T,'r--');
```

图 6-8 中，实线表示输出信号，虚线表示调制信号。从仿真结果看，输出信号和调制信号基本吻合，所设计网络可以很好地完成不同调制信号、不同调幅度的峰值检波，输出波形中的纹波可以通过低通滤波器滤除。

6.3　Hopfield 网络及 MATLAB 程序

Hopfield 网络最初是由美国物理学家 J.J Hopfield 于 1982 年首先提出的，故称为 Hopfield 网络。1985 年 Hopfield 本人和 Tank 用这种网络模型成功解决了优化组合问题中的具有典型意义的旅行商问题 TSP，在所有随机选择的路径中找到了十万分之一的最优路径，这是当时神经网络研究工作中具有突破性的进展。

Hopfield 网络作为一种全连接型的神经网络，曾经为人工神经网络的发展进程开辟了新的研究途径。它利用与阶层型神经网络不同的结构特征和学习方法，模拟生物神经网络的记忆机理，获得了令人满意的效果。Hopfield 网络有离散型和连续型两种形式。这里只介绍离散型 Hopfield 网络，因为离散型 Hopfield 网络的结构比较简单，在实际工程中的应用比较广泛。

6.3.1　Hopfield 网络介绍

Hopfield 网络结构如图 6-9 所示，它是一种单层反馈非线性网络，每一个节点的输出均反馈到其他节点的输入，整个网络都不存在自反馈。

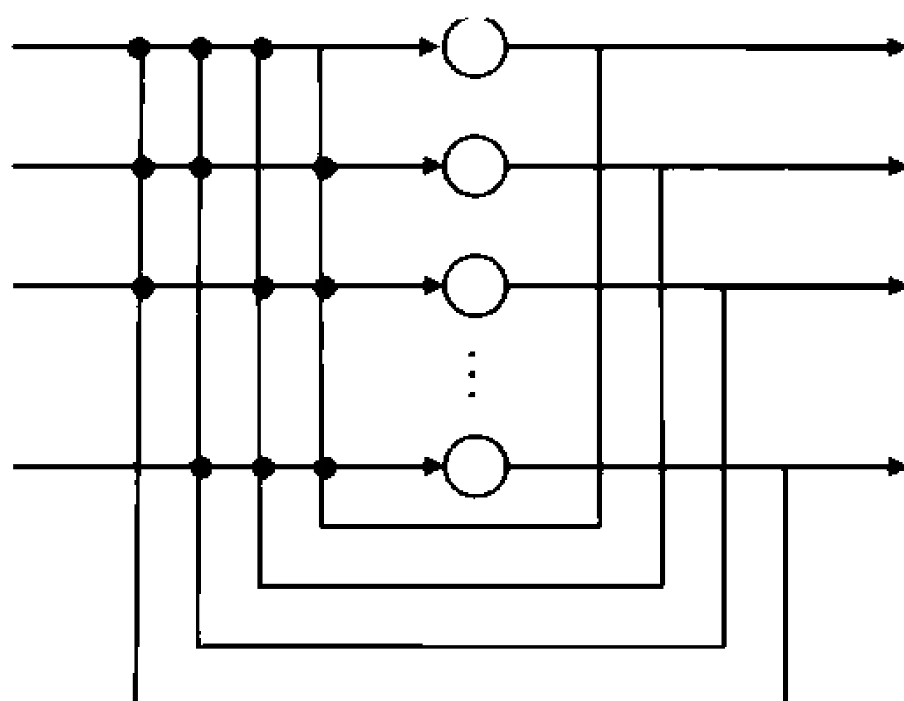


图 6-9　Hopfield 网络结构

J.J Hopfield 利用模拟电路（电阻、电容和运算放大器）实现了对网络节点（神经元）的描述，如图 6-10 所示。

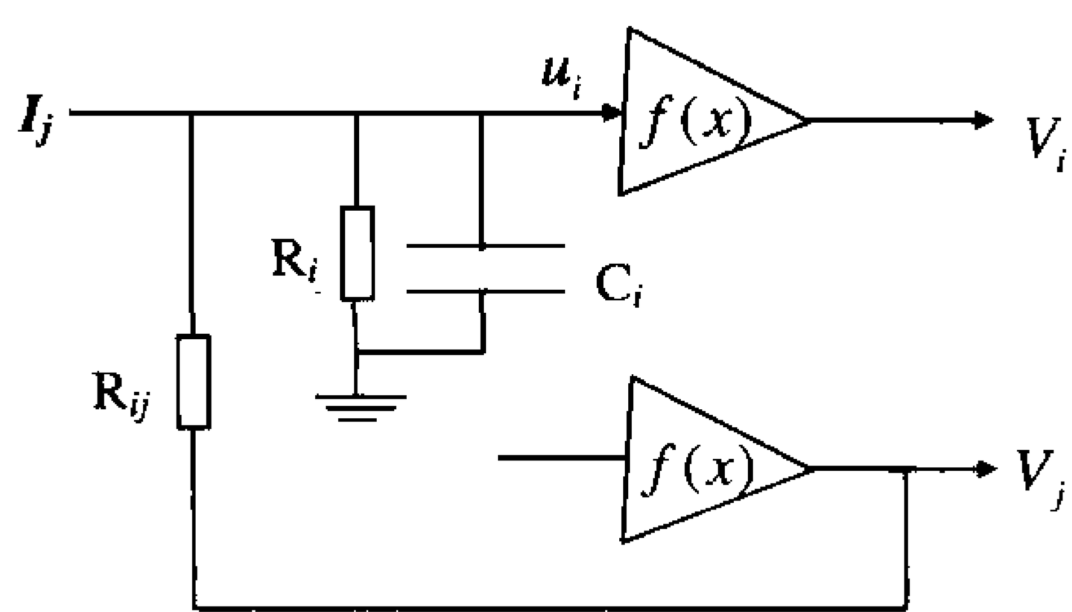


图 6-10　Hopfield 神经元的模拟电路

假设网络共有 n 个这样的神经元组成，可得出

$$\frac{dx_i}{dt}=-\frac{1}{\tau}x_i+\frac{1}{C_i}\sum_j w_{ij}y_j+\theta_j \tag{6-1}$$

$$y_i=f(x_i) \tag{6-2}$$

其中，$x_i=u_i$，$y_j=V_j$，$\tau=R_iC_i$，$\theta=I_j/C_i$，$w_{ij}=1/R_{ij}$，$w_{ii}=0$，$w_{ij}=w_{ji}$，且

$\frac{1}{R_i'}=\frac{1}{R_i}\sum_j\frac{1}{R_{ij}}$。传递函数 $f(x)$ 为 S 函数。

由此可见，R_i 与 C_i 的并联模拟了生物神经元的时间常数，w_{ij} 模拟了神经元间的突触特性即权值，运算放大器模拟了神经元的非线性特性，偏置电流 I_j 相当于阈值。

Hopfield 网络是渐近稳定的，随着时间的推移，网络状态向能量减小的方向移动，稳定平衡状态就是能量的极小点。因此，如果网络的初始状态在稳定平衡状态，则其状态不变；否则，网络需要运动到稳定平衡状态。

6.3.2 Hopfield 网络的学习

网络的学习过程实际上就是权值调整过程，Hopfield 网络的学习目的就是调整连接权值，以使得网络的稳定平衡状态就是所要求的状态。

Hopfield 网络常采用的学习算法是 Hebb 学习规则，即权值调整规则为：若第 i 个和第 j 个神经元同时处于兴奋状态，那么它们之间的连接增强，权值增大。

$$\Delta w_{ij}=ay_iy_j \quad a>0 \tag{6-3}$$

假设要求网络有 p 个正交稳态 $\boldsymbol{V}^s=\left(V_1^s,V_2^s,\cdots,V_n^s\right)$，$s=1,2,\cdots,p$，则

$$w_{ij}=\sum_{s=1}^{p}V_i^sV_j^s \tag{6-4}$$

若增加新的稳态 $\boldsymbol{V}^{p+1}$，则

$$w_{ij}^p=w_{ij}+V_i^{p+1}V_j^{p+1} \tag{6-5}$$

6.3.3 Hopfield 网络的几个重要结论

（1）联想记忆功能。由于网络可以收敛于稳定状态，因此可用于联想记忆。若将稳态视为一个记忆，则由初始状态向稳态收敛的过程就是寻找记忆的过程，初始状态可认为是给定了部分信息，收敛过程可认为是从部分信息找到了全部信息，实现了联想记忆的功能。

（2）优化计算。若将稳态视为某一优化问题目标函数的极小点，则由初始状态向稳态收敛的过程就是优化计算过程。

（3）网络渐近稳定的前提是 $w_{ij}=w_{ji}$。

（4）网络的应用。Hopfield 网络多用于在控制系统的设计中求解约束优化问题，另外在系统辨识中也有应用。

6.3.4 Hopfield 网络的 MATLAB 程序

【例 6-3】含有两个神经元的 Hopfield 网络的设计实例。

其完整的 MATLAB 代码如下。

```
T=[1 -1;-1 1]';
%画出 Hopfield 网络的状态空间和两个稳定点
```

```
plot(T(1,:),T(2,:),'+');
axis([-1.1 1.1 -1.1 1.1]);
%建立 Hopfield 网络
net=newhop(T);
for i=1:20
    A={rands(2,1)};
    [Y,Pf,Af]=sim(net,{1 20},{},A);
    Rd=[cell2mat(A) cell2mat(Y)];
    St=cell2mat(A);
    hold on;
    %画出网络状态的变化轨迹
    plot(St(1,1),St(2,1),'*',Rd(1,:),Rd(2,:));
end
```

程序运行结果如图 6-11 所示，图中的“*”符号表示网络的各初始状态。

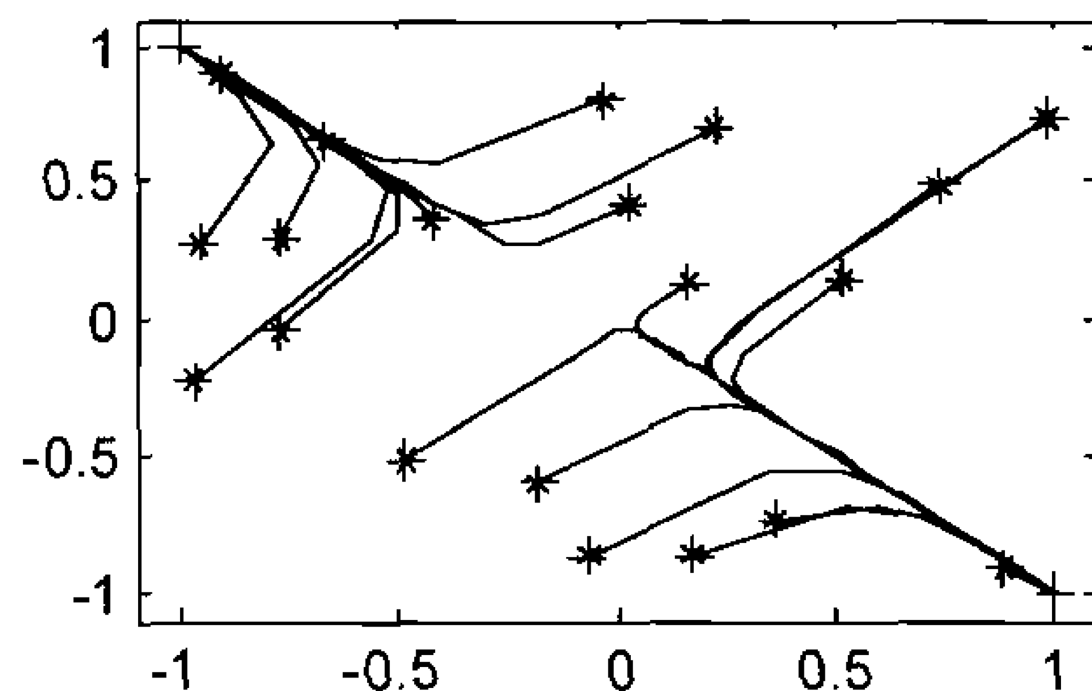


图 6-11　两神经元 Hopfield 网络显示结果

【例 6-4】设计一个含有三神经元的 Hopfield 网络。

其完整的 MATLAB 代码如下。

```
T=[1 -1 -1;-1 1 1]';
%画出 Hopfield 网络的状态空间和两个稳定点
plot3(T(1,:),T(2,:),T(3,:),'+');
axis([-1.1 1.1 -1.1 1.1 -1.1 1.1]);
axis manual;
set(gca,'box','on');
view([36.5 30]);
%建立 Hopfield 网络
net=newhop(T);
for i=1:20
    A={rands(3,1)};
    [Y,Pf,Af]=sim(net,{1 20},{},A);
    Rd=[cell2mat(A) cell2mat(Y)];
    St=cell2mat(A);
    hold on;
    %画出网络状态的变化轨迹
    plot3(St(1,1),St(2,1),St(3,1),'*',Rd(1,:),Rd(2,:),Rd(3,:));
end
```

程序运行效果如图 6-12 所示，图中“*”符号表示网络的各初始状态。

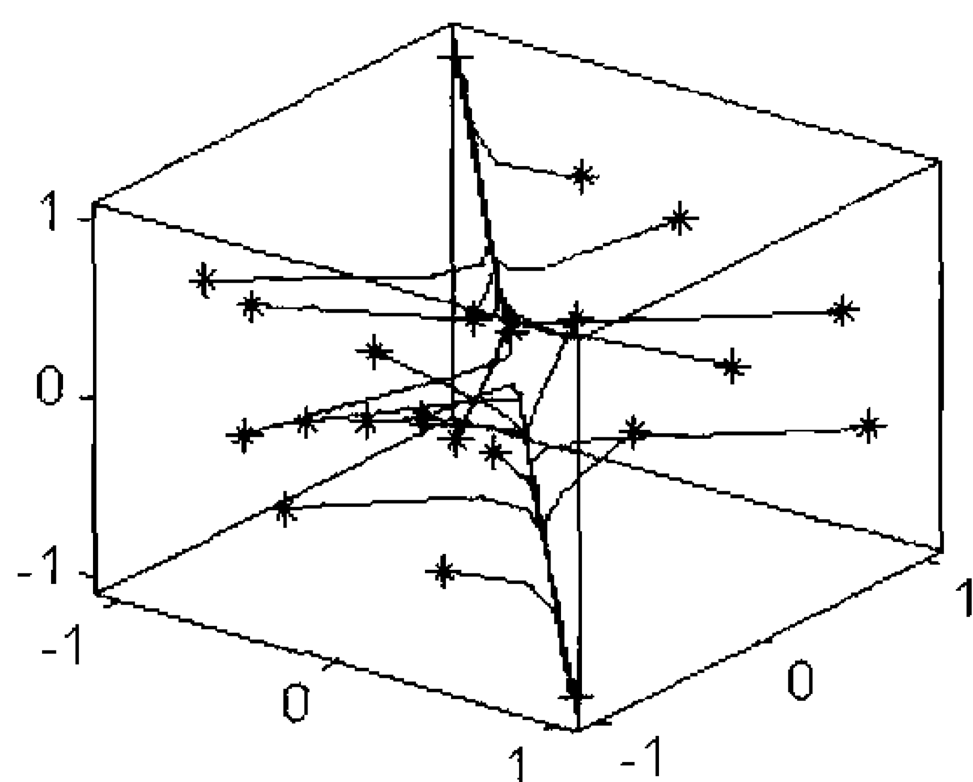


图 6-12　三神经元 Hopfield 网络显示结果

6.3.5　Hopfield 网络基于数学识别

字符识别是计算机模式识别的一个重要方面。作为字符识别的组成部分之一的数组识别在邮政、交通及商业票据管理方面有着极高的应用价值。目前有很多种方法用于字符识别，主要分为神经网络识别、概率统计识别和模糊识别等。近年来，神经网络识别技术发展较快，由于其本身具有自学习和自组织等优良特性而日益受到重视。目前，对印刷体的字符识别技术已经比较成熟了，而手写体字符由于书写者的因素，使得字符图像的随意性很大，比如笔画的粗细、字体的大小、手写体的倾斜度和局部扭曲等直接影响到字符的正确识别。因此，手写体字符的识别是字符识别领域最有挑战性的课题。

【例 6-5】设计一个 Hopfield 网络，使其具有联想记忆功能，能正确识别阿拉伯数字，当数字被噪声污染后仍可以正确地识别。

假设网络由 10 个初始稳态值 0~9 构成，即可以记忆 10 种数字。每个稳态由 10×10 的矩阵构成，该矩阵用于模拟阿拉伯数字点阵。所谓数字点阵，就是将数字划分成很多小方块，每一个小方块都对应着一部分数字。这里将数字划分成一个 10×10 方阵，其中，有颜色的方块用 1 表示，空白处用-1 表示，如图 6-13 所示。

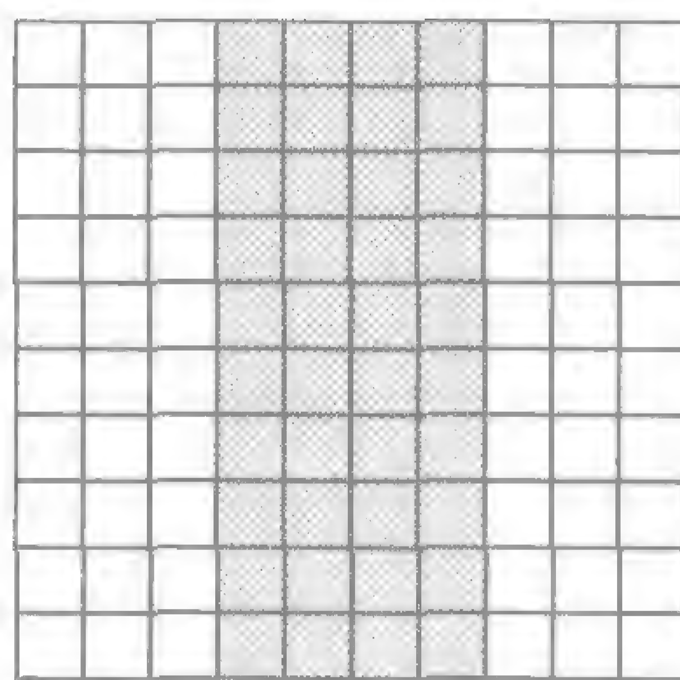


图 6-13　数字 1 的数字点阵

这样一来，就有

```
one=[-1 -1 -1 1 1 1 1 -1 -1 -1;-1 -1 -1 1 1 1 1 -1 -1 -1;-1 -1 -1 1 1 1 1 -1 -1 -1;-1 -1 -1 1 1 1 1 -1 -1 -1;
     -1 -1 -1 1 1 1 1 -1 -1 -1;-1 -1 -1 1 1 1 1 -1 -1 -1;-1 -1 -1 1 1 1 1 -1 -1 -1;-1 -1 -1 1 1 1 1 -1 -1 -1;
     -1 -1 -1 1 1 1 1 -1 -1 -1;-1 -1 -1 1 1 1 1 -1 -1 -1];
two=[1 1 1 1 1 1 1 1 -1 -1;1 1 1 1 1 1 1 1 -1 -1;-1 -1 -1 -1 -1 -1 1 1 -1 -1;-1 -1 -1 -1 -1 -1 1 1 -1 -1; 1 1
     1 1 1 1 1 1 -1 -1;1 1 1 1 1 1 1 1 -1 -1;1 1 -1 -1 -1 -1 -1 -1 1 1;-1 -1 -1 -1 -1 -1 1 1 -1 -1; 1 1 1 1 1
     1 1 1 -1 -1;1 1 1 1 1 1 1 1 -1 -1];
```

由于篇幅问题，这里只列出 1 和 2 的点阵表示形式。利用这两个向量构成一个训练样本，并由此构建一个 Hopfield 网络。

```
T=[one;two]';
net=newhop(T);
```

下面给出一个受噪声污染的数字 2 的点阵 No2。

```
No2={[1 1 1 -1 1 1 -1 1 -1 -1;1 1 1 1 1 1 1 1 -1 -1;-1 1 -1 1 -1 -1 1 1 -1 -1;-1 -1 1 -1 1 -1 1 1 -1 -1;1 1
    1 1 1 1 1 1 -1 -1;1 1 1 1 1 1 1 1 -1 -1;1 1 -1 -1 -1 -1 -1 -1 -1 -1;1 1 -1 -1 -1 -1 -1 -1 -1 -1;1 1 1 -1
    1 1 1 1 -1 -1;1 1 1 -1 1 1 1 1 -1 -1]'};
```

接下来尝试利用刚刚创建的 Hopfield 网络将受到噪声污染的数字 2 识别出来。

```
tu2=sim(net,{1,5},{},No2);
tu2{3}'
```

结果为

```
ans =
    1 1 1 1 1 1 1 1 -1 -1; 1 1 1 1 1 1 1 1 -1 -1; -1 -1 -1 -1 -1 -1 1 1 -1 -1; -1 -1 -1 -1 -1 -1 1 1 -1 -1;1 1
    1 1 1 1 1 1 -1 -1; 1 1 1 1 1 1 1 1 -1 -1; 1 1 -1 -1 -1 -1 -1 -1 -1 -1;-1 -1 -1 -1 -1 -1 -1 -1 1 1; 1 1 1 1
    1 1 1 1 -1 -1; 1 1 1 1 1 1 1 1 -1 -1;
```

结果和数字 2 的正常点阵是一致的，说明网络从受到污染的数字 2 的点阵中识别出了数字 2，此网络是有效的。

6.4　联想记忆

联想记忆（Associative Memory，AM）是神经网络理论的一个重要组成部分，也是神经网络用于智能控制、模式识别和人工智能等领域的一个重要功能。它主要利用神经网络的良好容错性，能使不完整的、污损的、畸变的输入样本恢复完整的原型，适于识别、分类等用途。Hopfield 网络模拟了生物神经网络的记忆功能，也常常被称为联想记忆网络。

6.4.1　联想记忆的基本概念

人类具有联想的功能，可以从一种事物联系到其相关的事物或其他事物。人工神经网络是对生物神经网络的模拟，也具有联想的功能。人工神经网络的联想就是指系统在给定一组刺激信号的作用下，该系统能联想出与之相对应的信号。联想记忆的过程就是信息的存取过程。数字计算机的信息存取方式是按地址存取方式（Addressable Memory），即一组信息对应着一定的存储单元。神经网络信息的存取（或记忆与联想）是以基于内容的存取原理为基础的，记忆地址通过记忆内容的部分描述来识别。所以这里所谓的联想记忆也称为基于内容的存取（Content-addressed Memory），信息被分布于生物记忆的内容之中，而不是某个确定的地址。

联想记忆网络是通过神经元之间的权重学习规则来调整神经元间的权重的，从而得到各事物间的联系。各神经元间的权重共同表现为神经网络的联想记忆功能。因此，联想记忆的神经网络是由突触权重和连接结构来对信息进行记忆。这种分布式能存储较多的模式，能够在一定的程度上恢复残缺的不完整信息。因而，它可以应用在图像复原、模式识别等领域，并具有两个比较突出的优点。

- 信息的存储是按内容存储记忆的（Content Addressable Memory，CAM），而传统的计算机是基于地址存储的。
- 信息的存储是分布的，而不是集中的。

1．联想记忆的分类

从作用方式来看，联想记忆分为线性联想与非线性联想；从状态来看，又可分为静态联想与动态联想。但在通常情况下，人们把联想记忆分为自联想与异联想。Hopfield 网络属于自联想。

1）自联想记忆（Auto-associative Memory）

自联想能将网络中输入模式映射到存储在网络中不同模式中的一种模式。联想记忆网络不仅能将输入模式映射为自己所存储的模式，而且还能对具有缺损或噪声的输入模式有一定的容错能力。

设在学习过程中给联想记忆网络存入 M 个样本：$\left(X^i\right)$，$i=1,2,\cdots,M$ 。若给联想记忆网络加以输入 $X'=X^m+V$ ，其中 X^m 是 M 个学习样本之一，V 是偏差项（可代表噪声、缺损与畸变等），通过自联想记忆网络的输出为 X^m，即使之复原（比如：破损照片→完整照片）。

一般情况下，自联想的输入与输出模式具有相同的维数。

2）异联想记忆（Hetero-associative Memory）

最早的异联想网络模型是 Kosko 的双向联想记忆神经网络。异联想网络在受到具有一定噪声的输入模式激发时，能通过状态的演化联想到原来样本的模式对。

假定两组模式对之间有一定的对应关系，$X^i \rightarrow Y^i$（如：某人照片→某人姓名），$i=1,2,\cdots,M$ 。若给联想记忆网络加以输入 $X'=X^m+V$ ，比如 X' 为某人的破损照片，通过异联想记忆网络的输出为 Y'，即得到某人的姓名。

异联想的输入模式维数与输出模式一般不相等。异联想可以由自联想通过映射得到。

2．联想记忆的工作过程

联想记忆的工作过程分为两个阶段：一是记忆阶段，也称为存储阶段或学习阶段；二是联想阶段，也称为恢复阶段或回忆阶段。

1）记忆阶段

在记忆阶段就是通过设计或学习网络的权值，使网络具有若干个稳定的平衡状态，这些稳定的平衡状态也称为吸引子（Attractor），吸引子有一定的吸引域（Basin of Attraction）。吸引子的吸引域就是能够稳定该吸引子的所有初始状态的集合，吸引域的大小用吸引半径来描述，吸引半径可定义为：吸引域中所含所有状态之间的最大距离或吸引子所能吸引状态的最大距离。

吸引子也就是联想记忆的网络能量函数的极值点；记忆过程就是将要记忆和存储的模式设计或训练成网络吸引子的过程。

2）联想阶段

联想过程就是给定输入模式，联想记忆网络通过动力学的演化过程达到稳定状态，即收敛到吸引子，回忆起已存储模式的过程。

吸引子的数量代表着联想记忆网络的记忆容量（Memory Capacity）或存储容量（Storage Capacity），存储容量就是在一定的联想出错概率容限下，网络中存储互不干扰样本的最大数目。存储容量与联想记忆的允许误差、网络结构、学习方式以及网络的设计参数有关。简单来说，一定的网络其吸引子越多，则网络的存储容量就越大。吸引子具有一定的吸引域，吸引域是衡量网络容错性的指标：吸引域越大，网络的容错性能越好，或者说网络的联想能力就越强。

6.4.2 Hopfield 联想记忆网络

如前所述，Hopfield 网络是一个神经动力学系统，具有稳定的平衡状态，即存在着吸引子，因而 Hopfield 网络具有联想记忆功能。将 Hopfield 网络作为联想记忆网络需要设计或训练网络

的权值，使吸引子存储记忆模式。比如，现在欲存储 m 个 n 维的记忆模式，那么就要设计网络的权值使这 m 个模式正好是网络能量函数的 m 个极小值。常用的设计或学习算法有：外积法（Outer Product Method）、投影学习法（Projection Learning Rule）、伪逆法（Pseudo Inverse Method）以及特征结构法（Eigen Structure Method）等。

现考虑离散 Hopfield 网络的联想记忆功能。设网络有 N 个神经元，每个神经元均取 1 或-1 二值，则网络共有 2^N 个状态，这 2^N 个状态构成离散状态空间。设在网络中存储 m 个 n 的记忆模式（$m<n$）

$$\boldsymbol{U}_k=[u_1^k,u_2^k,\cdots,u_i^k,\cdots,u_n^k]^{\mathrm{T}}，\quad k=1,2,\cdots,m；\quad i=1,2,\cdots,n；\quad u_i^k\in\{-1,1\} \tag{6-6}$$

采用外积法设计网络的权值使这 m 个模式是网络 2^n 个状态空间中的 m 个稳定状态，即

$$w_{ij}=\frac{1}{N}\sum_{k=1}^{m}u_i^k u_j^k \qquad i,j=1,2,\cdots,n \tag{6-7}$$

式中，$1/N$ 为调节比例的常量，这里取 $n=N$ 。考虑到离散 Hopfield 网络的权值满足条件 $w_{ij}=w_{ji}$，$w_{ii}=0$，则有

$$w_{ij}=\begin{cases}\dfrac{1}{n}\sum\limits_{k=1}^{m}u_i^k u_j^k, & j\neq i\\ 0, & j=i\end{cases} \tag{6-8}$$

将式（6-8）用矩阵形式表示，则有

$$\boldsymbol{W}=\frac{1}{n}\left(\sum_{k=1}^{m}\boldsymbol{U}_k\boldsymbol{U}_k^{\mathrm{T}}-m\boldsymbol{I}\right) \tag{6-9}$$

式中，$\boldsymbol{I}$ 为 $n\times n$ 的单位矩阵。

以上是离散 Hopfield 网络的存储记忆过程，下面再看其联想回忆过程。从所记忆的 m 个模式中任选一模式 $\boldsymbol{U}_l$，经过编码可使其元素取值为 1 和-1。设离散 Hopfield 网络中神经元的偏差均为零。将模式 $\boldsymbol{U}_l$ 加到该离散 Hopfield 网络，假定记忆模式矢量彼此是正交的（这是个特例，容易检验），则网络的状态为

$$\boldsymbol{U}_i^{\mathrm{T}}\boldsymbol{U}_j=\begin{cases}0 & (j\neq i)\\ n & j=i\end{cases} \qquad i,j=1,2,\cdots,m \tag{6-10}$$

$$\boldsymbol{W}\boldsymbol{U}_l=\frac{1}{n}\left(\sum_{k=1}^{m}\boldsymbol{U}_l\boldsymbol{U}_k^{\mathrm{T}}-m\boldsymbol{I}\right)\boldsymbol{U}_l=(n-m)\boldsymbol{U}_l \tag{6-11}$$

状态的演化为：Sgn($\boldsymbol{W}\boldsymbol{U}_l$)=Sgn($(n-m)\boldsymbol{U}_l$)=$\boldsymbol{U}_l$，可见网络稳定在模式 $\boldsymbol{U}_l$。

例如，对于两个记忆模式［1,-1,1］和［-1,1,-1］（这是一个记忆模式矢量非正交的例子）。按式（6-8）设计网络权值为

$$W = \frac{1}{3}\begin{bmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{bmatrix}$$

可见该权值满足离散 Hopfield 网络的条件。现将［1,-1,1］作为网络的输入，则有

$$Wy_1 = \frac{1}{3}\begin{bmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 4 \\ -4 \\ 4 \end{bmatrix}$$

$$\text{Sgn}[Wy_1] = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = y_1$$

可见状态［1,-1,1］为网络的稳定状态，即网络记住了该状态。同样，对状态向量［-1,1,-1］而言，有

$$Wy_2 = \frac{1}{3}\begin{bmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} -4 \\ 4 \\ -4 \end{bmatrix}$$

$$\text{Sgn}[Wy_2] = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = y_2$$

可见状态［-1,1,-1］也为网络的稳定状态，即网络也记住了该状态。

6.4.3 联想记忆网络的运行步骤

第一步：设定记忆模式。将欲存储的模式进行编码，得到取值为 1 和-1 的记忆模式（$n > m$）：$U_k = [u_1^k, u_2^k, \cdots, u_i^k, \cdots, u_n^k]^{\mathrm{T}}$； $k = 1,2,\cdots,m$ 。

第二步：设计网络的权值。

$$w_{ij} = \begin{cases} \dfrac{1}{N}\displaystyle\sum_{k=1}^{m} u_i^k u_j^k, & j \neq i \\ 0, & j = i \end{cases}$$

，其中 w_{ij} 是神经元 j 到 i 的突触权值，一旦计算完毕，突触权值将保持不变。

第三步：初始化网络状态。将欲识别模式 $U' = [u_1', u_2', \cdots, u_i', \cdots, u_n']^{\mathrm{T}}$ 设为网络状态的初始状态，即 $v_i(0) = u_i'$， $v_i(0)$ 是网络中任意神经元 i 在 $t = 0$ 时刻的状态。

第四步：迭代收敛。根据公式

$$v_i(t+1) = \text{Sgn}\left[\sum_{j=1}^{N} w_{ij} x_j(x)\right], \quad t = t+1$$

随机地更新某一神经元的状态，反复迭代直至网络中所有神经元的状态不变为止，假设此时的$t=T$。

第五步：网络输出。这时的网络状态（稳定状态）即为网络的输出$y=v_i(T)$。

以上的第一步和第二步是联想记忆网络的记忆过程，第三步至第五步所组成的迭代过程是联想记忆网络的联想过程。

对于以上所介绍的 Hopfield 联想记忆网络，需要做几点说明。

（1）以上所介绍的 Hopfield 联想记忆网络的激励函数为符号函数，即神经元的状态取 1 和 −1 的情况。对于联想记忆网络的激励函数为阶跃函数，即神经元的状态取 1 和 0 时，相应的公式有所变化。设在网络中存储 m 个 n 维的记忆模式（$n>m$）

$$\boldsymbol{U}_k=[u_1^k,u_2^k,\cdots,u_i^k,\cdots,u_n^k]^{\mathrm{T}} \tag{6-12}$$

式中，$k=1,2,\cdots,m$；$i=1,2,\cdots,n$；$u_i^k\in\{0,1\}$

对$u_i^k\in\{-1,1\}$的情况，前面已经讨论过了，而当$u_i^k\in\{0,1\}$时，可以进行一个变换，即使得$(2u_i^k-1)\in\{-1,1\}$，所以用$(2u_i^k-1)$代换前面公式中的u_i^k即可

$$w_{ij}=\begin{cases}\dfrac{1}{n}\sum\limits_{k=1}^{m}(2u_i^k-1)(2u_j^k-1), & j\neq i\\ 0, & j=i\end{cases} \tag{6-13}$$

矩阵形式为

$$\boldsymbol{W}=\frac{1}{n}\left(\sum_{k=1}^{m}(2\boldsymbol{U}_k-\boldsymbol{E}_{n\times1})(2\boldsymbol{U}_k^{\mathrm{T}}-\boldsymbol{E}_{1\times n})-m\boldsymbol{I}\right) \tag{6-14}$$

式中，$\boldsymbol{E}_{n\times1}=(1,1,\cdots,1)^{\mathrm{T}}$，$\boldsymbol{E}_{1\times n}=(\boldsymbol{E}_{n\times1})^{\mathrm{T}}$，$\boldsymbol{I}$表示单位阵。其联想回忆过程同上面讨论过的。

例如，对于两个记忆模式［1,0,1］和［0,1,0］，按式（6-14）设计网络权值为

$$\boldsymbol{W}=\frac{1}{3}\begin{bmatrix}0 & -2 & 2\\ -2 & 0 & -2\\ 2 & -2 & 0\end{bmatrix}$$

可见该权值满足离散 Hopfield 网络的条件。现将［1,0,1］作为网络的输入，则有

$$\boldsymbol{W}\boldsymbol{y}_1=\frac{1}{3}\begin{bmatrix}0 & -2 & 2\\ -2 & 0 & -2\\ 2 & -2 & 0\end{bmatrix}\begin{bmatrix}1\\ 0\\ 1\end{bmatrix}=\frac{1}{3}\begin{bmatrix}2\\ -4\\ 2\end{bmatrix}$$

$$f(\boldsymbol{W}\boldsymbol{y}_1)=\begin{bmatrix}1\\ 0\\ 1\end{bmatrix}=\boldsymbol{y}_1$$

可见状态［1,0,1］为网络的稳定状态，即网络记住了该状态。同样，对状态向量［0,1,0］而言，有

$$Wy_2 = \frac{1}{3}\begin{bmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} -2 \\ 0 \\ -2 \end{bmatrix}$$

$$f(Wy_2) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = y_2$$

可见状态［0,1,0］也为网络的稳定状态，即网络也记住了该状态。

(2)Hopfield 联想记忆网络的记忆容量α反映了所记忆的模式个数m和神经元的数目N之间的关系：$\alpha = m/N$。记忆m个模式所需的神经元数$N = m/\alpha$，连接权值数目为$(m/\alpha)^2$，若α增加一倍，连接权值数目降为原来的 1/4，这是一对矛盾。在技术实现上也是很困难的。Hopfield 通过实验和理论研究发现，网络大约能存储$0.15N$个记忆模式。近年来，人们对网络的记忆容量进行了很多研究，发现了一些提高记忆容量的方法，比如利用具有优化选择非单调性的 Hopfield 联想记忆网络，可使其记忆容量达到$0.4N$。

(3) Hopfield 联想记忆网络存在伪状态（Spurious States），伪状态是指除记忆状态之外的网络多余的稳定状态。从以上对 Hopfield 联想记忆网络的分析可见，要构成一个对所有输入模式都很合适的 Hopfield 联想记忆网络是很不容易的，需要满足的条件是很苛刻的。所记忆的模式m过大，权值矩阵W中会存在若干个相同的特征值；所记忆的模式m小于神经元的数目N，权值矩阵 W 中会存在若干个 0，构成所谓的零空间。因此，零空间存在于网络中，零空间是 Hopfield 网络的一个固有特性，即 Hopfield 联想记忆网络不可避免地存在着伪状态。

(4) 网络的平衡稳定点不可以任意设置，也没有一个通用的方法事先知道平衡稳定点。如果使用外积学习算法，当记忆模式矢量彼此正交时，可以联想出正确结果，而当它们不正交时，就必须满足一定条件，才能进行正确联想。一个判别样本是否稳定的简单有效的方法由以下定理给出。

定理：已经m个n维的记忆模式$U_k = [u_1^k, u_2^k, \cdots, u_i^k, \cdots, u_n^k]^T$，$k = 1,2,\cdots,m$；$i = 1,2,\cdots,n$。则$U_j$为稳定样本的充要条件是：对一切$i = 1,2,\cdots,n$，都有

$$u_i^j \sum_{k=1}^{m}\left(u_i^k \sum_{\substack{l=1 \\ l \neq i}}^{n} u_l^j u_l^k \right) > 0$$

6.4.4 联想记忆网络的改进

在讨论联想记忆网络的改进前，首先要明确对联想记忆网络的基本要求，这就是：

(1) 联想记忆网络须具有较大的记忆容量。

(2) 网络联想记忆须具有一定的容错性，即吸引子要有一定的吸引域。

(3) 是技术可实现的。

这三者是一个统一的整体，而容错性则是其核心，没有容错性就谈不上联想，容错性的优劣是由各吸引子吸引域的大小与形状所决定的。

联想记忆网络的根本问题之一就是除存在记有记忆样本的吸引子之外，还有“多余”的稳定状态即伪状态存在，伪状态的存在影响到联想记忆网络的容错性。若能减小甚至消除伪状态的吸引域，就可提高联想记忆网络的容错性，增大记忆容量。

为此可从系统的三个方面考虑：

（1）调节系统的动力学特性，但为了利用系统的并行性，同步动力学是必要的。

（2）改变网络权值矩阵的设计或学习算法，如除采用外积法外还可考虑采用投影学习法、伪逆法或特征结构法等。这些方法都是人为设定的，因有较好的性能而常被采用。但从优化的角度看，它们并非是最优的。要人为设定一种适合于各种样本集的最佳方案是很困难的，应充分发挥神经网络可学习性这一优点，用训练方法使之自动找出最优或较优解。这些方法着眼于使样本被“记住”及提高记忆容量上，而忽略了对其容错性的考虑。

（3）修改神经元的激励函数，对于激励函数常用的是符号函数，也可采用连续单调函数作为激励函数，即采用连续的 Hopfield 网络，连续型 Hopfield 网络对于高维系统建立的是高阶非线性微分方程，是非常难以模拟和实现的。而取二值或多值的离散型 Hopfield 网络，通过合适长度的二进制编码，可以任意精度表示任意实数，而且所占用的存储空间较小。

6.4.5　Hopfield 神经网络应用于联想记忆的 MATLAB 程序

【例 6-6】Hopfield 神经网络用于联想记忆。假设希望构造一个联想记忆模型，这个模型能够识别出如图 6-14 所示的 4 个数字。

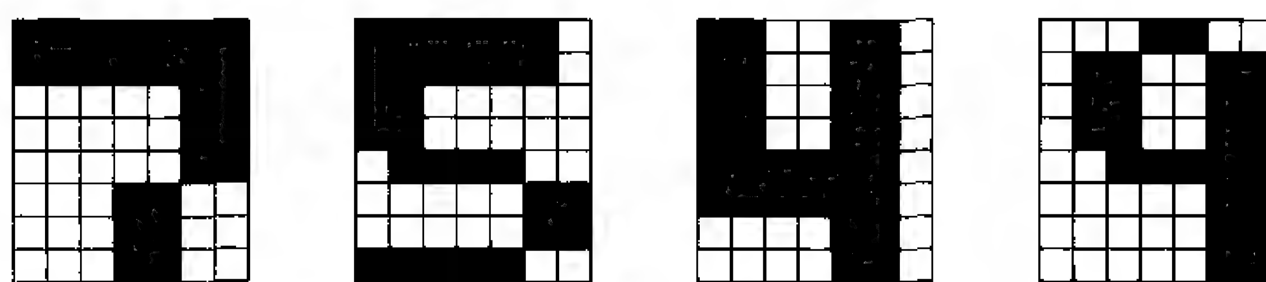


图 6-14　待识别数字的 8×7 二值化图像

（1）问题分析。

图 6-14 所示二值化数字图像为二维图像，若以 0 表示数字笔画划过的小方块，以 1 表示数字笔画未划过的小方块，则 Hopfield 网络需要 56 个神经元表示各方块的状态，同时要求目标向量是一维的，用一维向量表示。

例如，数字“7”的目标向量如下。

```
T7=[0 0 0 0 0 0 0
    0 0 0 0 0 0 0
    1 1 1 1 1 0 0
    1 1 1 1 1 0 0
    1 1 1 1 1 0 0
    1 1 1 0 0 1 1
    1 1 1 0 0 1 1
    1 1 1 0 0 1 1]';
```

（2）设计 Hopfield 神经网络的 MATLAB 程序。

```
T7=[0 0 0 0 0 0 0
    0 0 0 0 0 0 0
    1 1 1 1 1 0 0
    1 1 1 1 1 0 0
    1 1 1 1 1 0 0
```

```
    1 1 1 0 0 1 1
    1 1 1 0 0 1 1
    1 1 1 0 0 1 1]';
>> clear all;
T7=[0 0 0 0 0 0 0
    0 0 0 0 0 0 0
    1 1 1 1 1 0 0
    1 1 1 1 1 0 0
    1 1 1 1 1 0 0
    1 1 1 0 0 1 1
    1 1 1 0 0 1 1
    1 1 1 0 0 1 1]';
T5=[0 0 0 0 0 0 1
    0 0 0 0 0 0 1
    0 0 1 1 1 1 1
    0 0 1 1 1 1 1
    1 0 0 0 0 1 1
    1 1 1 1 1 0 0
    1 1 1 1 1 0 0
    0 0 0 0 0 1 1]';
T4=[0 0 1 1 0 0 1
    0 0 1 1 0 0 1
    0 0 1 1 0 0 1
    0 0 1 1 0 0 1
    0 0 0 0 0 0 1
    0 0 0 0 0 0 1
    1 1 1 1 0 0 1
    1 1 1 1 0 0 1]';
T9=[1 1 1 0 0 1 1
    1 0 0 1 1 0 0
    1 0 0 1 1 0 0
    1 0 0 1 1 0 0
    1 1 0 0 0 0 0
    1 1 1 1 1 0 0
    1 1 1 1 1 0 0
    1 1 1 1 1 0 0]';
%形成总的目标向量
T=[T7 T5 T4 T9];
%设计 Hopfield 网络
net=newhop(T);
```

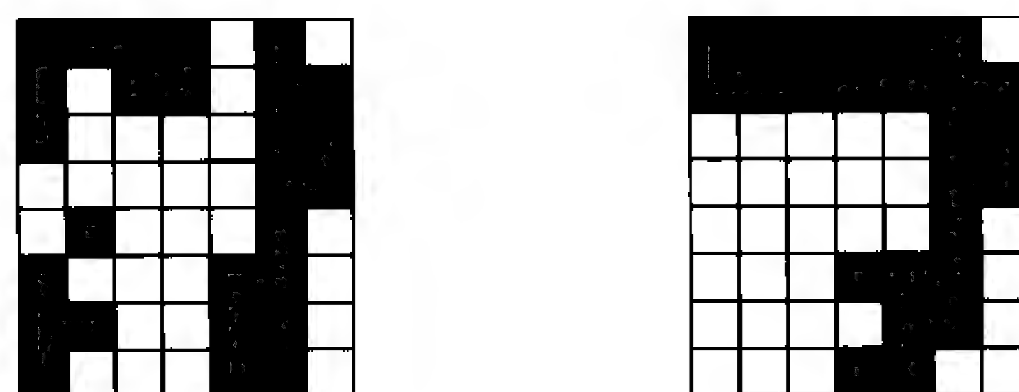
（3）Hopfield 神经网络的 MATLAB 仿真程序设计。

以加噪并且产生畸变的数字“7”作为测试对象，对所设计的 Hopfield 神经网络进行仿真。

```
T7=[0 0 0 0 0 0 0
    0 0 0 0 0 0 0
    1 1 1 1 1 0 0
    1 1 1 1 1 0 0
    1 1 1 1 1 0 0
```

```
1 1 1 0 0 1 1
1 1 1 0 0 1 1
1 1 1 0 0 1 1]';
subplot(121);
%绘制测试样本二值化图像
figt(T7);
%网络仿真
y=sim(net,1,[ ],T7);
%二值化
y=y>0.5;
subplot(122);
%仿真输出二值化图像
figt(y)
```

仿真结果如图 6-15 所示。从结果上可以看出，受到噪声影响并且严重畸变的数字“7”，通过网络仿真后的输出已非常接近网络存储的对应于“7”的样本模式。



(a) 测试样本　　(b) 仿真结果

图 6-15　测试样本和仿真结果的二值化图像

源代码中的 figt(t)是绘制测试样本二值化图像的自定义函数，其源代码如下。

```
%绘制测试样本二值化图像的自定义函数 figt(t)
function figt(t)
hold on
axis square
for j=1:8
     for i=1:7
          if t((j-1)*7+i)==0
               fill([i i+1 i+1 i],[9-j,9-j,10-j,10-j],'k')
          else
               fill([i i+1 i+1 i],[9-j,9-j,10-j,10-j],'w')
          end
     end
end
hold off
```

6.5　回归 BP 网络及 MATLAB 应用

误差反射传播 BP 算法是前向网络学习算法中应用最为广泛的算法，因此，一些研究者尝试将 BP 算法中采用的梯度下降法推广到回归网络中，由此产生了回归 BP（Recurrent BP）网络。

6.5.1 回归 BP 网络概述

回归 BP 网络同时具有反馈和前馈机制，这就意味着在网络的一个训练周期中，网络的输出同时反馈给网络的输入神经元作为网络的外部输入。如图 6-16 所示为一个典型的三层回归 BP 网络。

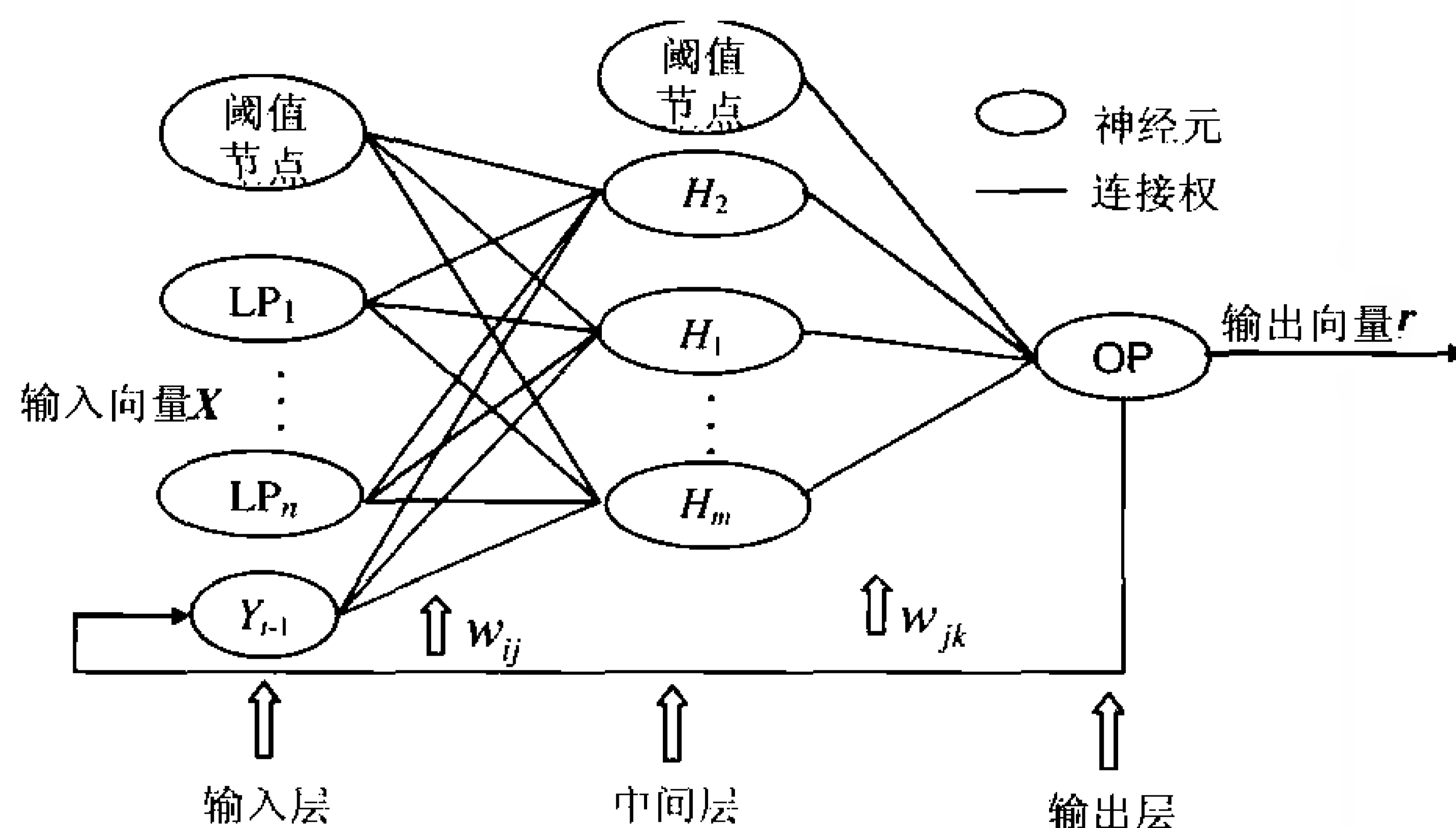


图 6-16 回归 BP 网络

在图 6-16 中，输入层有一个 n 维的输入向量和一个阈值节点，该节点的值是固定的，这个值的存在保证了网络的收敛特性。中间层有 m 个神经元和一个阈值节点。相邻两层的所有神经元采用全连接的方式相连。输入层的神经元输入/输出关系可以表示为

$$I_i^F = O_i^F = X_i, i = 1,2,\cdots,n$$
$$O_i^F = f(I_i^F) = f(X_i)$$

其中，F 表示输入层，输入神经元和隐含神经元之间实现加权连接。即如果信号从第 i 个神经元传递到 j 个神经元，则信号需要乘上两个神经元之间的连接权值 w_{ij}。令 O_i 表示第 i 个神经元的输出，第 j 个神经元的输入则为 $O_i w_{ij}$。对第 j 个神经元的输入进行求和，可得

$$I_j^H = \sum_{i=1}^{n_1} O_i^F w_{ij}^F + \theta_j^H, \quad j = 1,2,\cdots,m$$

式中，θ_j 为阈值项，H 表示中间层。这种加法操作是通过中间层的处理器实现的，实现加法的过程就是激发神经元的过程。由于神经元的权值和输入可以取正值，也可以取负值，因此对神经元的激发也可能产生正值、负值和零值中的任意数据。中间层的激发函数为

$$O_j^H = f(I_j^H) = f\left(\sum_{i=1}^{n_1} O_i^F w_{ij}^F + \theta_j^H\right) \quad j = 1,2,\cdots,m$$

对于输出层 Y，它的第 k 个神经元接收了中间层第 j 个神经元的输出信号，经过加权后，作为自己的输入信号，可以得到同上面类似的结论。

$$I_k^Y = \sum_{j=1}^{m_1} O_j^H W_{jk}^H + \theta_k^Y, \quad k = 1,2,\cdots,g$$

$$O_k^Y = f(I_k^Y) = f\left(\sum_{j=1}^{m_1} O_j^H W_{jk}^H + \theta_k^Y\right),\quad k = 1,2,\cdots,g$$

回归 BP 网络的传递函数见下式，这是一种非常典型的函数，具有很多优点。它是连续可导的，输出也是连续的并且位于 0～1 中。如图 6-17 所示，β 为函数的斜率。

$$O_k^Y = f(net) = \frac{1}{1+\exp(-\beta \times net)},\quad \beta > 0$$

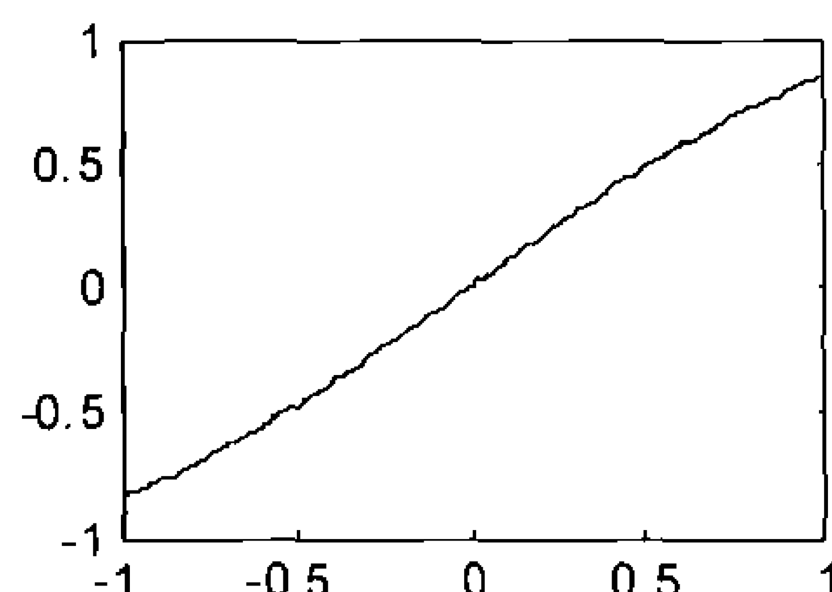


图 6-17　回归 BP 网络传递函数曲线（β=1）

6.5.2　回归 BP 网络在房价的应用

房价及其变化趋势是很多人关注的焦点，也是制定合理的购房策略的基础。现以某城市的房价预测为例，展开基于回归 BP 网络的预测研究。

一般来说，房价受宏观调控政策、房产和人口等多方面的影响，可以将房价写为如下的函数模型。

$$RHP = f(Q_d, Q_s, t) = f(x_i, y_i, z_i, v_i, t)$$

其中，RHP 表示房价，$t = 1,2,\cdots,n$，$i = 1,2,\cdots,m$。Q_d 表示在 t 时间段中对房子的总体需求，Q_s 表示在 t 时间段中房子的总体供应。x 表示宏观经济变量，如 GDP 等。y 表示与买方相关的变量，如家庭平均收入等。z 表示与人口统计相关的变量，如人口比例、新婚人口数目等。v 表示与卖方相关的变量，如建房费用和地皮价格等。x、v、y 和 z 是完全无关的变量。

由于房价受多方面因素的影响，在建立一个房价预测模型时，需要综合考虑多种因素。而 BP 算法在处理多输入的非线性系统的过程中表现出了相当强的能力，所以这里采用 BP 算法。之所以采用递归 BP 网络，是因为房价是一种时间序列信号，目前的数据可能对以后的数据产生重要的影响。

由于 MATLAB 神经网络工具箱中没有为递归 BP 网络提供专门的工具，所以这里只给出基于递归 BP 网络进行房价预测的有关思想，具体算法读者可参考相关资料，并利用 MATLAB 的数学计算功能实现。

6.6　BSB 模型及其应用

盒中脑（Brain-State-in-a-Box，BSB）神经网络模型首先是由 Anderson 等人于 1977 年提出的，Golden 等人对该模型进行了深入的研究。BSB 模型是一种节点之间存在横向连接和节点自反馈的单层网络，可用做自联想最邻近分类器，并可存储任何模拟向量模式。

6.6.1 BSB 神经网络介绍

BSB 网络模型可用如下方程描述。

$$X(k+1)=g(X(k)+\alpha WX(k)) \quad k=0,1,2\cdots$$

初始条件为 $X(0)=X_0$，其中 $X(k)=(x_1(k),x_2(k),\cdots,x_n(k))\in R^n$ 表示 k 时刻的状态向量，参数 α 是一正值，用于控制层内反馈的大小。$W\in R^{n\times n}$ 为对称的权矩阵，传递函数 g 的第 i 个坐标通常为以下形式

$$g_i(X)=\begin{cases}1 & x_i\geqslant 1\\ x_i & |x_i|<1\\ -1 & x_i\leqslant -1\end{cases}$$

随着时间的推移，每个状态 x_i 逐渐趋近于 ±1。实际上，当系统达到某个平衡态后，状态 $(x_1,x_2,\cdots,x_n)$ 进入由 $(\pm1,\pm1,\cdots,\pm1)$ 构成的盒子某一角。

6.6.2 BSB 的应用

神经网络工具箱中没有为 BSB 网络提供专门的函数工具，因此，无法利用神经网络工具箱中的函数创建、训练并应用网络。但是，Hugh Pasika 于 1997 年基于 MATLAB 平台开发了 BSB 网络的实现函数。代码如下。

```
function C=BSB(X,beta,multi)
%function C=BSB(X,beta)
% This m-file duplicates the Brain State in a Box Experiment.
% X    -input vector
% beta -feedback factor
%C        -number of iterations requried for convergence
% Hugh Pasika 1997
hold on
flag-0;
x=x(:);
c=2;   %C is a general purpose counter
W=[.035 -.005;-.005 .035];
% set axes
set(gca,'XLim',[-1 1]);
set(gca,'XLim',[-1 1]);
%plot first point
plot(x(1),x(2),'ob');
Og=x';
% plot center lines
plot([0,0],[1,-1],'+');
plot([1,-1],[0,0],'+');
% label plot
set(gca,'YTick',[-1 1]);
set(gca,'XTick',[-1 1]);
while flag<1
```

```
        y=x+beta*W*x;
        x=(y(:,:)<-1)*(-1)+(y(:,:)>1)+(y(:,:)>-1 & y(:,:)<1).*y;
        u(c,:)=x';
        c=c+1;
        if u(c-1,:)==u(c-2,:),
            flag=10;
            c=c-3;
        end
    end
    u=u(2:c+1,:);
    Og
    plot([Og(1,1) u(1,1)],[Og(1,2) u(1,2)],'-b')
    plot(u(:,1),u(:,2),'ob')
    plot(u(:,1),u(:,2),'-b')
    drawnow
    fprintf(1,'It took %g iteration for a stable point to be reached.\n\n',c);
    set(gca,'Box','on')
    hold off
```

读者可以将上述代码保存为一个名为 BSB.m 的文件，存储到 MATLAB 的根目录下。然后在 MATLAB 主窗口中将 Current Directory 当前目录修改为 MATLAB 的根目标，在命令行窗口中输入"help bsb"后按回车键，出现如下的帮助信息。

```
help bsb
  function C=BSB(X,beta)
    This m-file duplicates the Brain State in a Box Experiment.
   X    -input vector
   beta -feedback factor
   C    -number of iterations requried for convergence
    Hugh Pasika 1997
```

帮助信息解释了函数各个参数的意义，其中

X：输入向量；

beta：反馈因子；

C：最大迭代次数。

下面给定输入向量和反馈因子，演示 bsb 函数的功能。令 x=[0.5;-0.6]，beta=0.5，c=100，c 用于限制迭代次数。迭代终止的充分条件是网络已经收敛或者网络的迭代次数达到最大值 c。在命令行窗口中输入以下代码后按回车键。

```
Og =
    0.5000    -0.6000
It took 133 iteration for a stable point to be reached.
x=[0.5;-0.6];
beta=0.5;
c=100;
BSB(x,beta,c)
ans=
   25
```

其中 Og 表示网络的初始输入值为［0.5,-0.6］。ans=25 表示网络经过 25 次迭代后，初始值就达到了箱子的一角［1,-1］，因为初始值与它的距离最小。输入向量的收敛轨迹如图 6-18 所示。

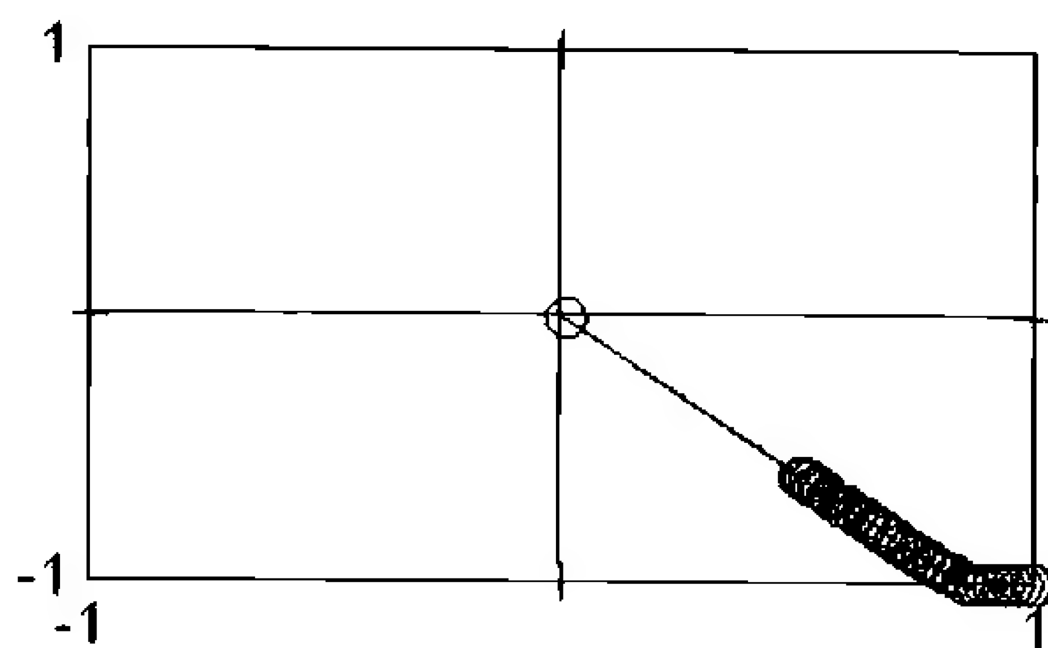


图 6-18　输入向量的收敛轨迹

第 7 章　图形用户接口

前面几章介绍了 MATLAB 神经网络工具箱的各种函数，这些函数是神经网络仿真程序设计的基础，可以给用户以充分的开发空间，按照自己构想设计各种神经网络。但对于程序设计或神经网络工具箱函数不是很熟悉的用户来说，要快捷、方便、正确地设计一个神经网络是非常困难的。

MATLAB 的神经网络工具箱 Neural Network Toolbox Version 4.x 提供了图形用户界面（Graph User Interface，GUI），从而使用户在图形界面上，通过与计算机的交互操作设计和仿真神经网络，使得神经网络的设计和仿真变得简单易学。

7.1　图形用户界面介绍

在 MATLAB 命令窗口（command window）中输入 nntool，即可打开“Network/Data Manager”（网络/数据管理器）窗口，如图 7-1 所示。



图 7-1　“Network/Data Manager”窗口

“Network/Data Manager”窗口有 7 个显示区域和 2 个按钮区：

（1）Inputs 区域：显示用户指定的输入向量变量名。

（2）Targets 区域：显示用户指定的目标向量变量名。

（3）Input Delay States 区域：显示用户指定的输入延迟参数变量名。

（4）Networks 区域：显示用户定义的网络名。

（5）Outputs 区域：显示网络的输出向量变量名。

（6）Errors 区域：显示网络的训练误差变量名。

（7）Layer Delay States 区域：显示用户指定的网络层延迟参数变量名。

（8）Networks and Data 按钮区。

- “Help”按钮：单击该按钮，弹出“Network/Data Manager Help”窗口，为用户使用 Network/Data Manager Help 提供帮助。

- "New Data…"按钮：单击该按钮，弹出"Create New Data"窗口，在该窗口可以定义各种数据类型的变量名和数据值（Value）。
- "New Network…"按钮：单击该按钮，弹出"Create New Network"窗口，在该窗口可以定义神经网络名称、神经网络类型及其网络对象和子对象属性参数等。
- "Import…"按钮：单击该按钮，弹出"Import or Load Network/Data Manager"窗口，可以通过该窗口从命令窗口或磁盘文件导入神经网络或数据。
- "Export…"按钮：单击该按钮，弹出"Export or Save from Network/Data Manager"窗口，可以将"Network/Data Manager"窗口的变量导出到命令窗口或存入磁盘文件中。
- "View"按钮：先选中显示区域的变量名或网络名，单击"View"按钮，则弹出一个新的窗口，在该窗口中显示选中的变量或网络的具体内容。
- "Delete"按钮：先选中显示区域的变量名或网络名，单击"Delete"按钮，则删除选中的变量或网络。

（9）Networks only 按钮区：先选中显示区域的网络名，单击该区域的任意一个按钮，则弹出一个新的窗口（"Network：网络名"），在该窗口中，可以查看网络的结构示意图和权值/阈值，设置网络的初始化值、训练参数、自适应调整参数和仿真参数，并可对定义的神经网络进行初始化、训练、自适应调整、仿真等。

7.2 图形用户应用示例

仍以例 4-9 的模式分类问题为例，将待分类模式重画于图 7-2 中。

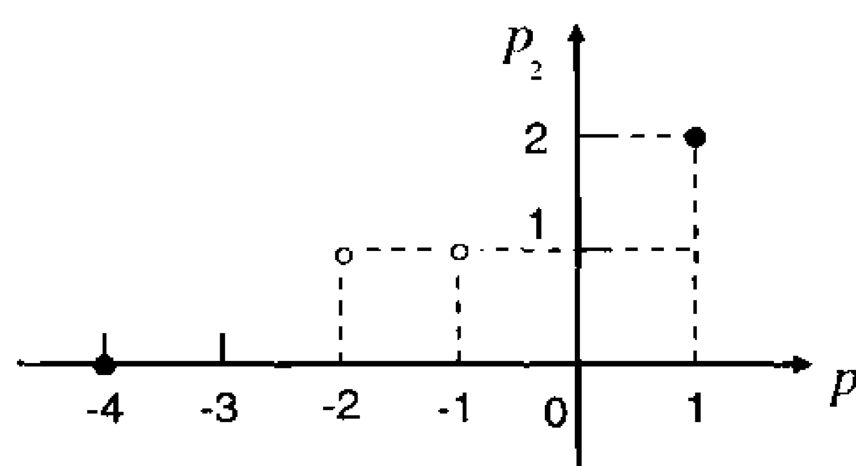


图 7-2 待分类模式

据例 4-9 的分析，网络结构重画于图 7-3 中。第 1 层有 5 个神经元，第 2 层有 1 个神经元。

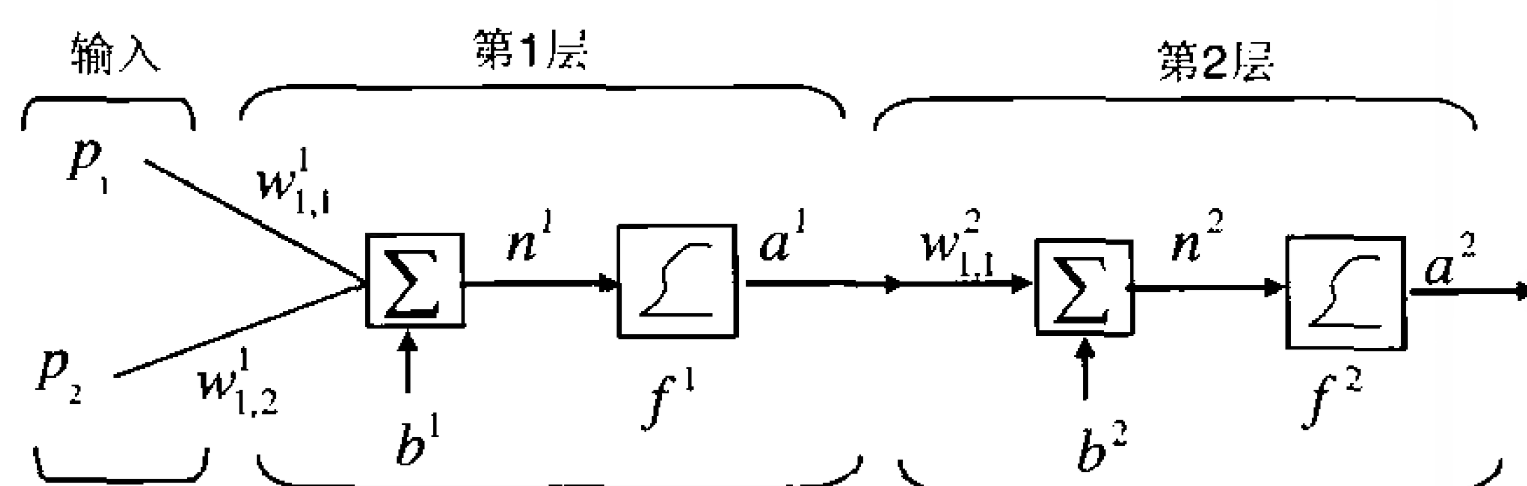


图 7-3 两层 BP 网络

训练样本集为

$$\boldsymbol{p}=\begin{bmatrix}1 & -1 & -2 & -4\\ 2 & 1 & 1 & 0\end{bmatrix},\quad \boldsymbol{t}=[0.2\quad 0.8\quad 0.8\quad 0.2]$$

用图形用户界面设计上述神经网络的具体方法如下。

（1）在 MATLAB 命令窗口中输入 nntool，打开"Network/Data Manager"窗口。

（2）创建神经网络：单击“New Network...”按钮，弹出“Create New Network”窗口，如图 7-4 所示。

① 输入网络名（Network Name）：Demonet。

② 选择网络类型（Network Type）：Feed-forward backprop。

③ 确定输入向量的取值范围（Input ranges）：[-4 1;0 2]。

④ 选择训练函数（Traning function）：TRAINLM。

⑤ 选择自适应调整学习函数（Adaption learning function）：LEARNGDM。

⑥ 选择误差性能函数（Performance function）：MSE。

⑦ 确定网络层数（Number of layers）：2。

⑧ 确定各网络层的属性（Properties for）。

layer1：神经元数（Number of neurons）为 5；

　　　　传输函数（Transfer Function）为 LOGSIG。

layer2：神经元数（Number of neurons）为 1；

　　　　传输函数（Transfer Function）为 LOGSIG。

⑨ 单击“View”按钮可以查看以上定义的网络结构，如图 7-5 所示。

⑩ 单击“Create”按钮，关闭“Create New Network”窗口，回到“Network/Data Manager”窗口，可以看到 Networks 区域显示出刚建立的网络名 Demonet，选中该网络名，单击该窗口的“View”按钮也可以查看到如图 7-5 所示的网络结构。

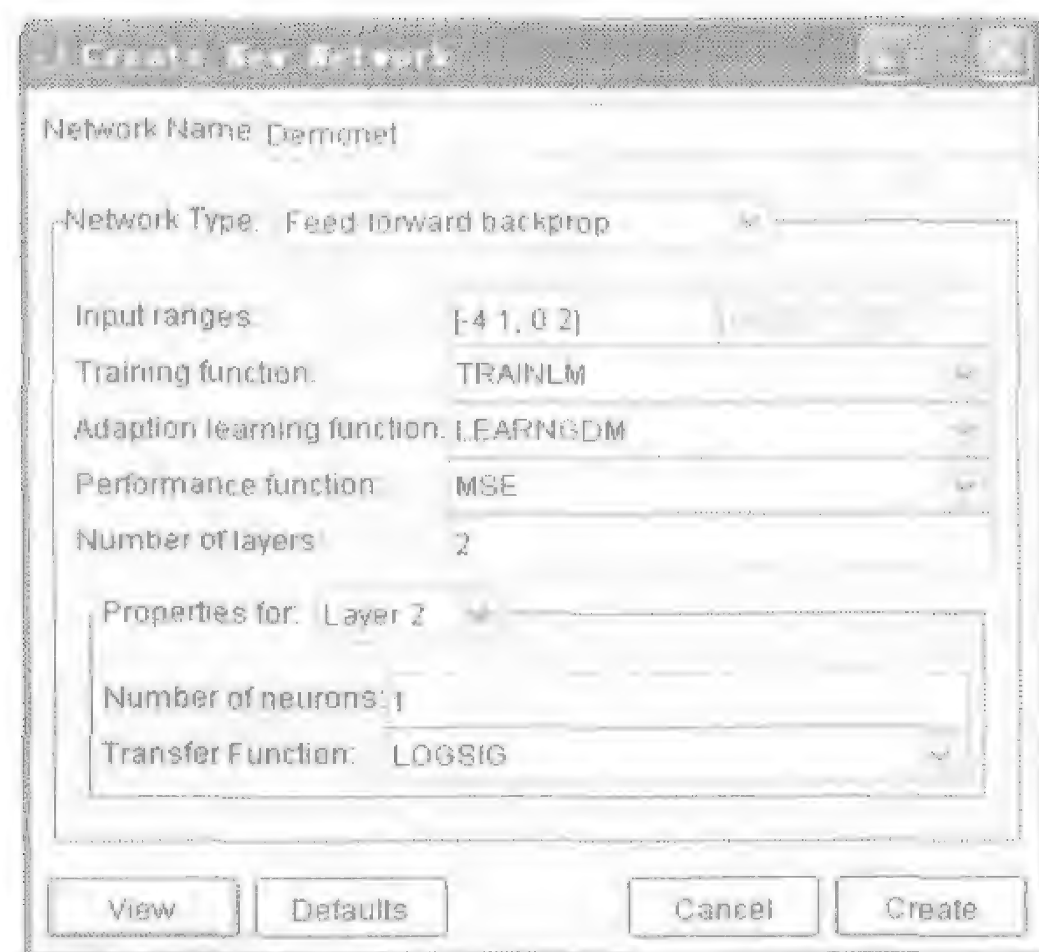


图 7-4　“Create New Network”窗口

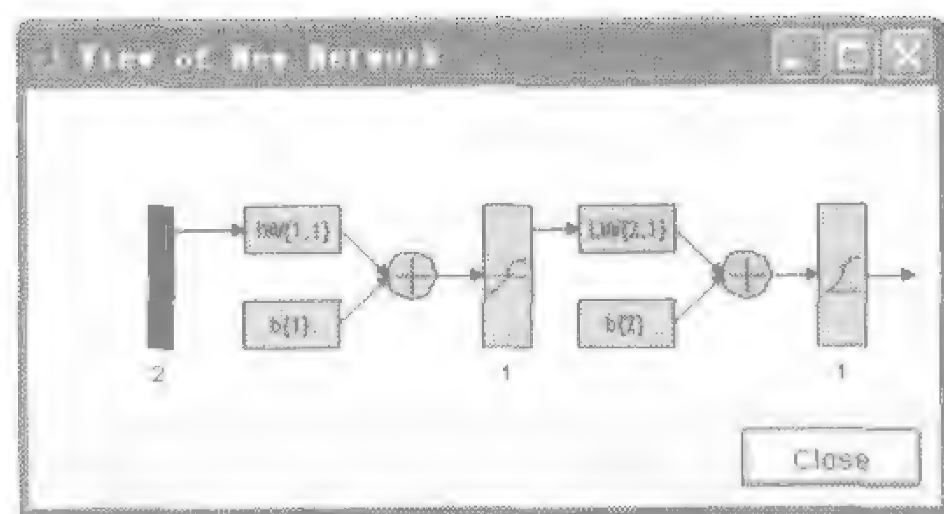


图 7-5　查看网络结构

（3）训练网络。

① 确定训练样本的输入向量。在“Network/Data Manager”窗口单击“New Data...”按钮，弹出“Create New Data”窗口，选择数据类型为 Inputs，输入向量名（Name）为 p，其值（Value）为[1 -1 -2 -4;2 1 1 0]，如图 7-6 所示。然后单击“Create”按钮，关闭“Create New Data”窗口，回到“Network/Data Manager”窗口。可以看到在 Inputs 区域显示出输入向量名 p，选中该输入向量名，单击该窗口的“View”按钮，弹出数据（Data）窗口，在该窗口可以查看到该输入向量的值，并可以修改数据值。

② 确定训练样本的目标向量。按照与输入向量同样的方法可以确定目标向量，只是选择数据类型为 Targets，输入向量名为 t，数据值为[0.2 0.8 0.8 0.2]。

③ 训练网络。在“Network/Data Manager”窗口选中网络名 Demonet，单击“Train...”按钮，则弹出“Network：Demonet”窗口，如图 7-7 所示。

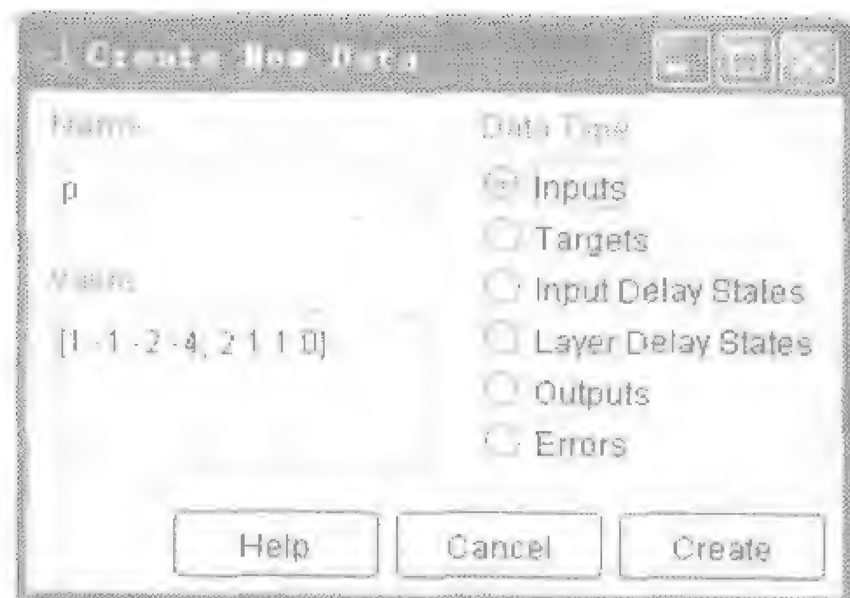


图 7-6 “Create New Data”窗口

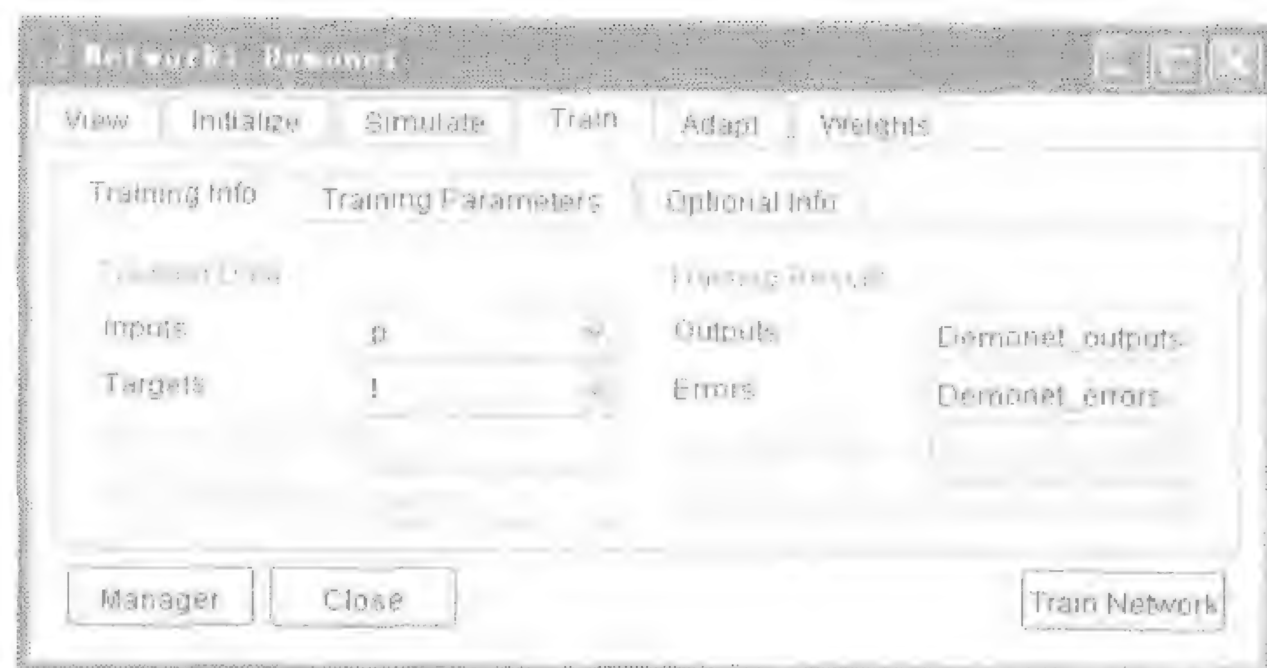


图 7-7 “Network：Demonet”窗口

可以看出，该窗口为一个多页面对话框，在 Train 页面有 3 个子页面。

- Training Info：在该子页面将训练数据（Training Data）的输入向量（Inputs）选择为 p，目标向量（Targets）选择为 t；训练结果（Training Results）的输出变量（Outputs）和误差性能变量（Errors）采用系统自动生成的 Demonet_outputs 和 Demonet_errors，当然它们也可以由用户重新定义。
- Training Parameters：在该子页面可以设置训练的各种参数，这要根据具体训练和学习函数进行确定，相关内容参看各神经网络模型的训练和学习算法。在此采用其默认值即可。
- Optional Info：该子页面用以确定在训练时是否采用确认样本和测试样本，本例均不采用。

以上过程完成后，单击该页面的“Train Network”按钮，开始训练，其训练过程如图 7-8 所示。

训练完成后，在 Network/Data Manager 窗口可以看到，在 Outputs 区域显示出输出变量名 Demonet_outputs，在 Errors 区域显示出误差性能变量名 Demonet_errors。选中变量名，单击该窗口的“View”按钮，则弹出数据（Data）窗口，在该窗口可以查看到该选中变量的具体数据。

（4）网络仿真。

在“Network/Data Manager”窗口选中网络名 Demonet，单击“Simulate...”按钮，弹出“Network：Demonet”窗口，显示 Simulate 页面，如图 7-9 所示。

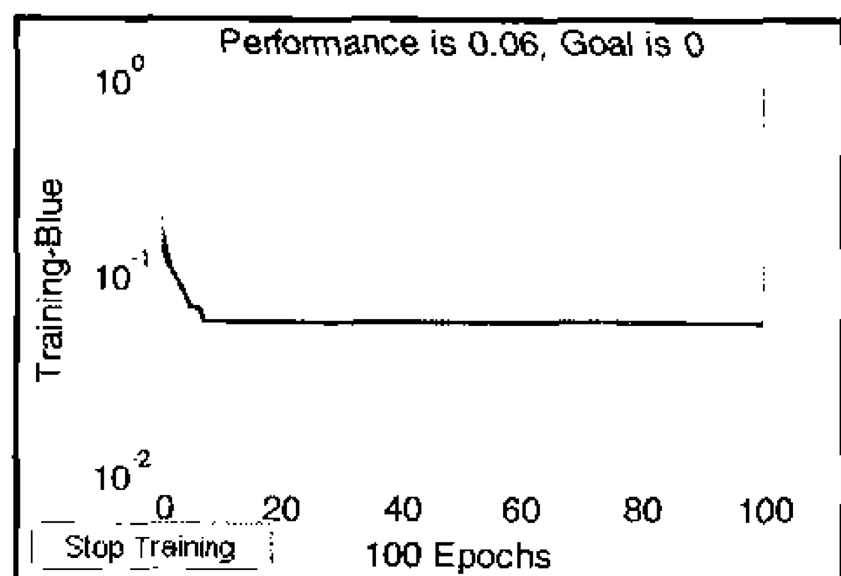


图 7-8 训练误差性能曲线

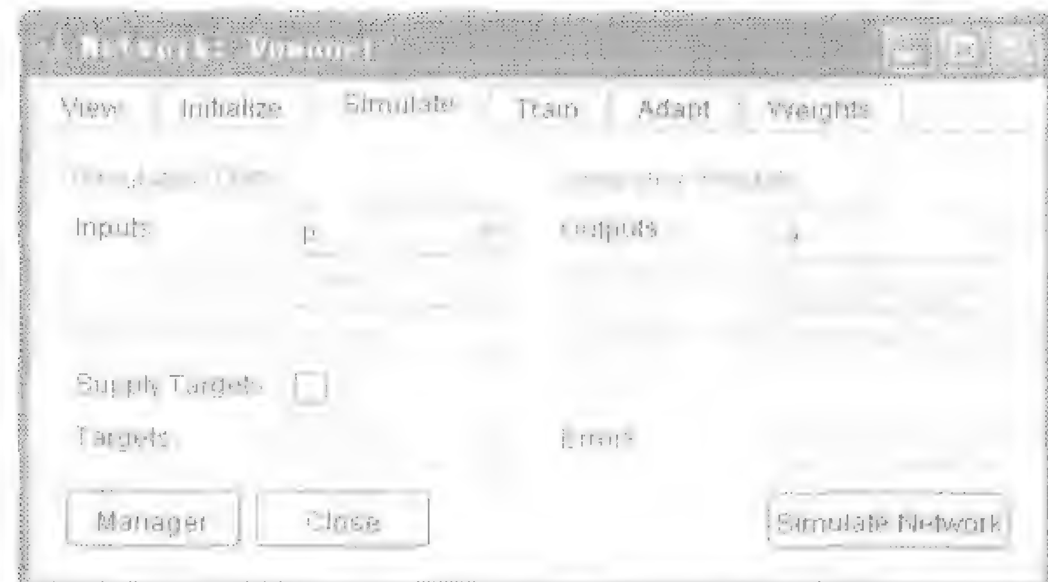


图 7-9 “Network:Demonet”窗口的 Simulate 页面

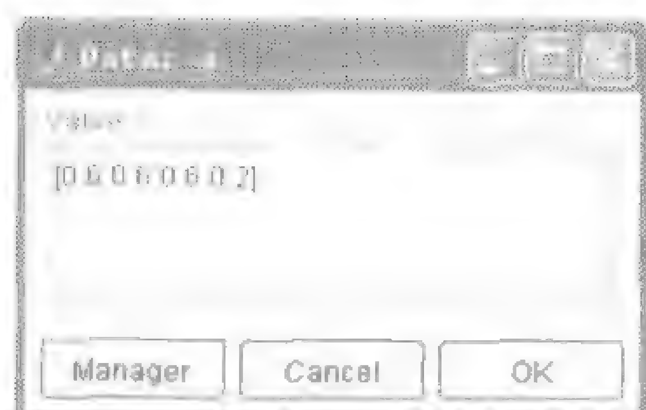


图 7-10 仿真结果数据

将仿真数据选择为 p，仿真结果选择为 a，单击“Simulate Network”按钮，则在“Network/ Data Manager”窗口的 Outputs 区域显示出输出变量名 a，选中该变量名，单击该窗口的 View 按钮，弹出数据（“Data：a”）窗口，在该窗口可以查看到仿真结果的具体数据，如图 7-10 所示。

可以看出，网络很好地完成了图 7-2 所示的两类模式分类问题。当然，可以用训练样本以外的数据进行仿真，此时，需要先在“Network/Data Manager”窗口建立仿真的输入向量，建立方法与建立训练样本的输入向量相同，然后在“Network：Demonet”窗口的 Simulate 页面选择该仿真的输入向量名，进行仿真。

7.3 数据操作

数据操作一方面指的是将 MATLAB 工作空间中的数据导入到 GUI 中，也可以将 GUI 中的数据变量导出到 MATLAB 工作空间中；另一方面是指数据的存储、删除、恢复和重新调用等。

7.3.1 数据从工作空间导入到 GUI

可以通过如下的方式将工作空间中的数据变量导入到 GUI 中。

首先，在 MATLAB 命令行中输入如下代码。

```
P=-1:0.15:1;
T=sin(pi*p);
```

这样一来，就得到了数据向量 ***P*** 和 ***T***，它们都由 41 个元素组成。回到“Network/Data Manager”窗口，单击其中的“Import...”按钮，得到如图 7-11 所示的对话框。

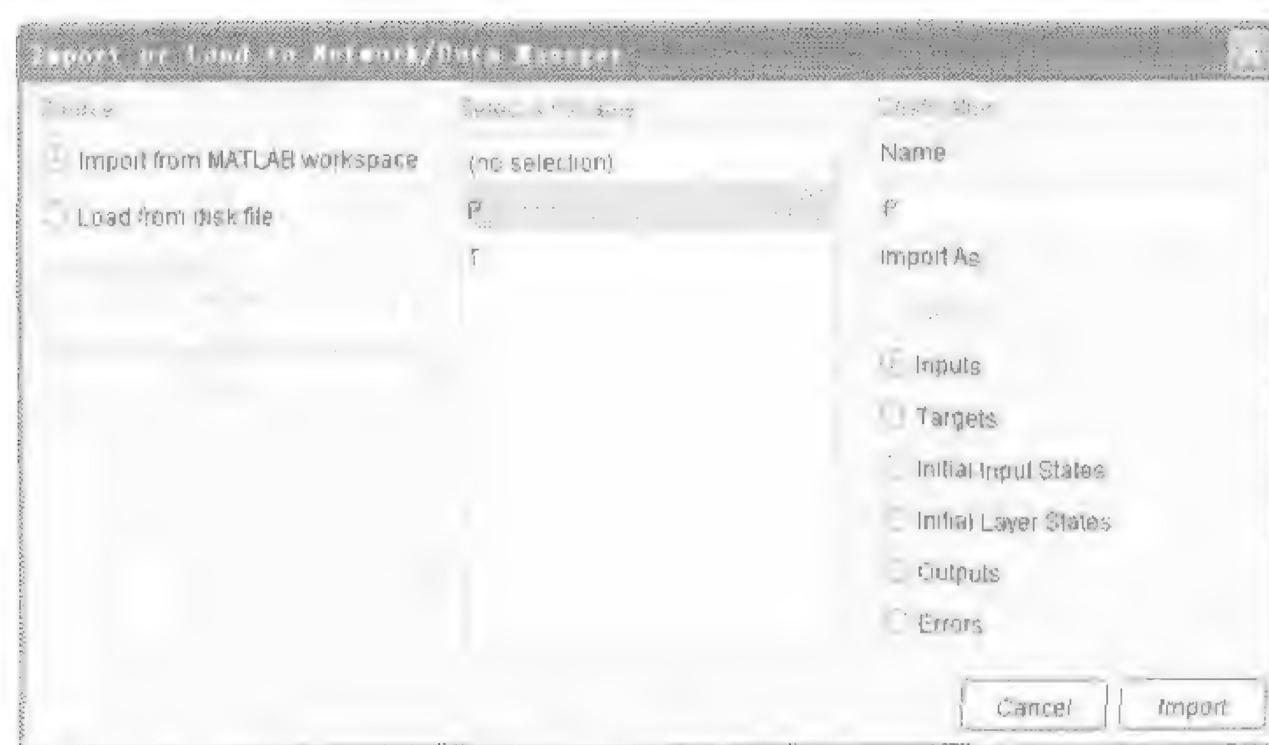


图 7-11 数据输入

选中此对话框左侧的 Source 域中的“Import from MATLAB workspace”单选钮，然后在其中的“Select a Variable”文本框中选中 P，在对话框右侧的“Name”文本框中输入 P，表示在 GUI 中该数据向量的名称亦为 P，然后选中“Inputs”单选钮，表示该数据向量是网络的输入向量。最后单击右下角的“Import”按钮，就可以将 ***P*** 作为网络的输入向量导入到 GUI 中。按照同样的方式，也可以将 ***T*** 作为网络的目标向量导入到 GUI 中。此时，新导入的 ***P*** 和 ***T*** 分别出现在“Network/Data Manager”窗口中的“Inputs”和“Targets”文本框中，分别表示神经网络的输入向量和目标向量。

此外，该对话框 Source 域中还提供了另外一个选项 Load from disk file，这表明还可以从磁盘中导入数据。

7.3.2 数据从 GUI 导出到工作空间

通过 GUI 得到的网络仿真结果、训练结果和误差等，都可以导出到 MATLAB 的命令行空间中，在命令行工作空间中可以利用这些数据进行一些其他操作。

以仿真结果的导出为例介绍数据的导出过程。仿真得到数据向量 ***a*** 后，回到“Network/Data Manager”窗口，在右侧的“Outputs”框中出现了 a 变量，如图 7-12 所示。

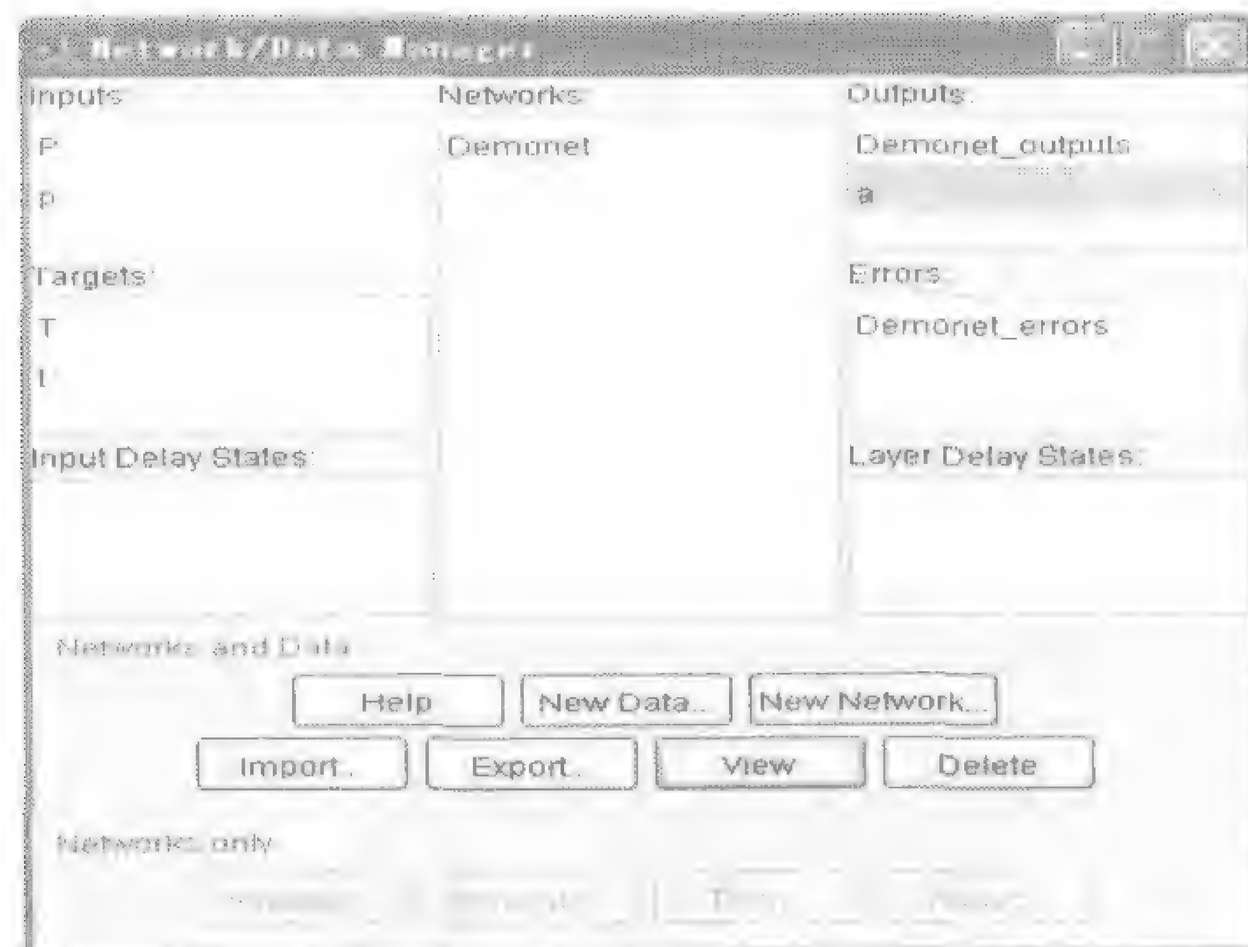


图 7-12 “Network/Data Manager”窗口

选中 a 变量，然后单击“View”按钮，可以看到该变量的元素数据，如图 7-13 所示。

单击“Export...”按钮，出现数据导出对话框，如图 7-14 所示。这里列出了 GUI 当前所有的数据变量。这里选中变量 a，并单击“Export”按

钮，将该对话框关闭，变量a就导出到命令行窗口中了。为了检验以上过程是否成功，在MATLAB命令行中输入 who 来检查此时工作空间中的所有变量，可得到以下结果。

```
Your variables are:
P   T   a
```

这表明变量 a 已经成功地导入到命令行空间中了。

可以按照同样的方式将设计好的网络导出到命令行空间中，导出成功后，在命令行窗口中输入 simubp，可得到网络的基本信息，如下。

```
simubp =
    Neural Network object:
    architecture;
    numInputs:1
  numLayers:2
  biasConnect:[1;1]
  inputConnect:[1;0]
  layerConnect:[0 0;1 0]
  outputConnect:[0 1]
  targetConnect:[0 1]
  numOutputs:1    (read-only)
  numTargets:1    (read-only)
  nunInputDelays:0    (read-only)
  numLayersDelays:0    (read-only)
  ……
```



图 7-13　变量显示



图 7-14　数据导出对话框

7.3.3　数据的存储和读取

通过 GUI 定义的变量、数据和网络可以存储到硬盘中，需要的时候可以再加载，这样就可以使得设计好的网络能够重复使用，而不用每次都要重新设计。下面以神经网络的存储和读取为例，介绍数据的存储和读取过程。

回到图 7-14 中，选中 a 项将其激活，然后单击右下角的“Save”按钮，出现“Save to a MAT file”对话框，如图 7-15 所示。

单击“保存”按钮，将网络以*.MAT 文件的形式存储在硬盘中，这里将文件名设定为 ab。

如果不小心将网络错误删除或者需要重新启动 MATLAB，我们还可以利用导出功能将网络文件导出到 GUI 中。下面介绍在重新启动 MATLAB 的情况下，如何将网络文件导出到 GUI 中。

重新启动 MATLAB 后，在命令行窗口输入 nntool 命令，启动“Network/Data Manager”窗口，单击“Import…”按钮，出现如图 7-16 所示的数据导入对话框。



图 7-15　“Save to a MAT file”对话框

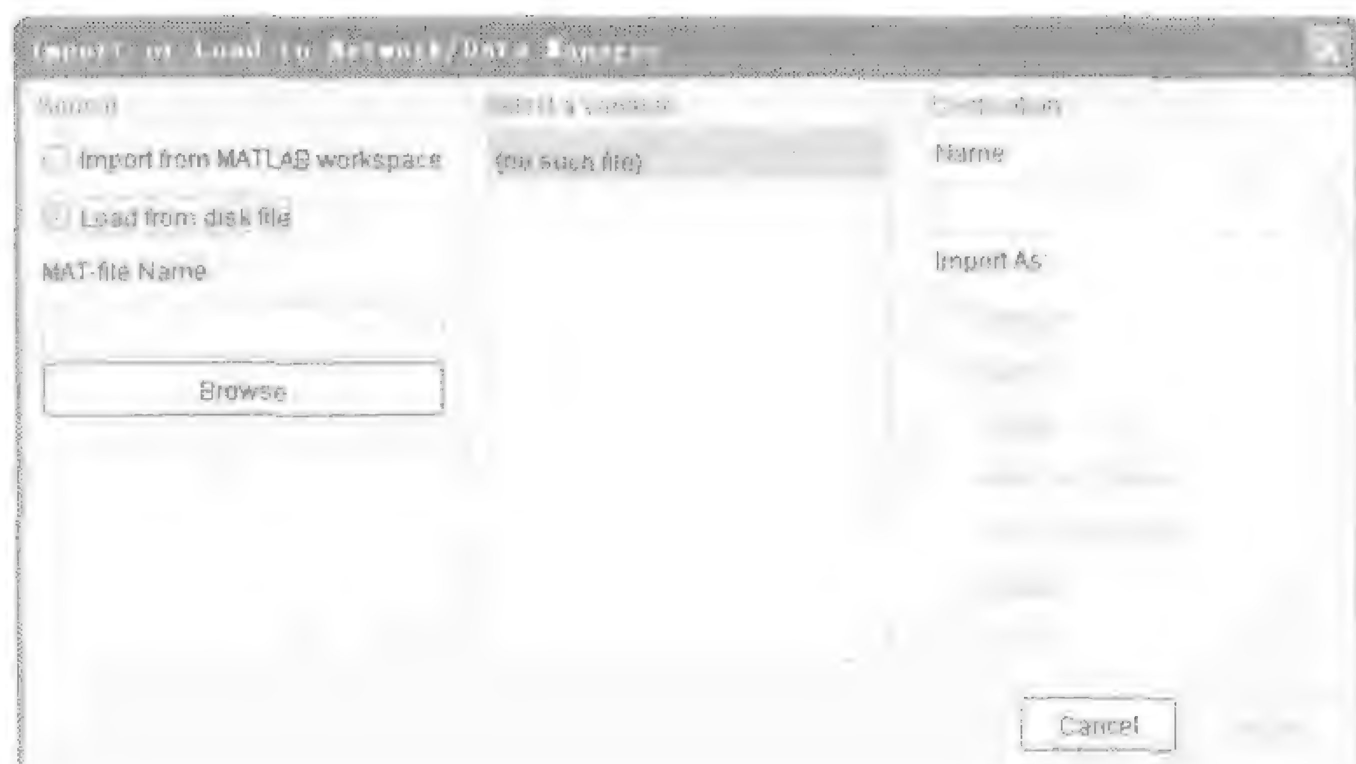


图 7-16　数据导入对话框

在 Source 域中选中“Load from disk file”单选钮，然后单击“Browse”按钮确定网络文件的来源，如图 7-17 所示。选中“ab.MAT”文件，单击“打开”按钮或者双击该文件，将文件导入到 GUI 中。



图 7-17　网络文件选择

单击“打开”按钮后，出现如图 7-18 所示的窗口。在“Select a Variable”框中选中“ab”，可以在“Name”文本框中输入网络在 GUI 中的名称，这里取原来的名称，也就是默认的名称“a”。MATLAB 有识别数据类型的能力，这里它已经识别出导入的数据类型为神经网络，“Import As”下的单选项中除了“Network”是无效的，其余的都是有效的，为可选。

最后单击位于窗口右下角的“Load”按钮，我们发现网络 a 已经在 GUI 中了，如图 7-19 所示。

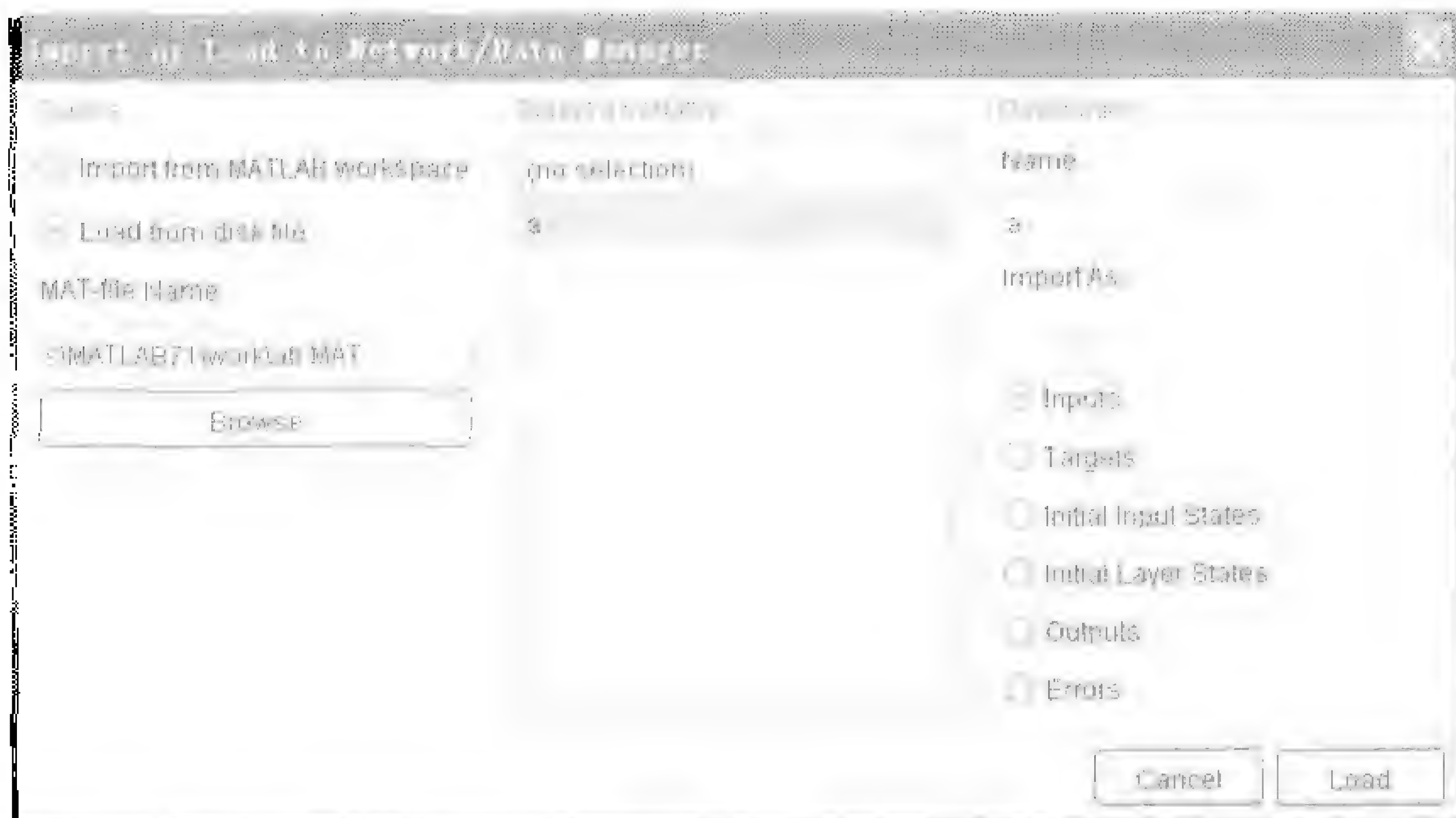


图 7-18　网络文件导入



图 7-19　导入网络 a 后的“Network/Data Manager”窗口

7.3.4　数据的删除

在通过 GUI 进行神经网络的设计分析时，有时候可能会出现误操作。例如，在设计网络输

入向量的时候出现了错误，但直到后来才发现，这时候就需要删除错误的输入向量，从而设计新的输入向量。

数据删除最简单的方法是强行关闭 MATLAB 窗口，或者到任务管理器中结束 MATLAB.EXE 进程。这种方法的优点是速度非常快，但必须要提醒读者的是，利用这种方法进行数据删除以前，必须将所有认为有用的数据变量保存起来，然后在重新启动 MATLAB 后，将它们一一导入到新的 GUI 中，这样才可以保证数据不丢失。

另外一种方法是在“Network/Data Manager”窗口中选中要删除的数据，如图 7-19 所示，选中了网络 a，然后单击“Delete”按钮，就可以将 a 删除了。

注意

删除 GUI 工作空间中的数据并不影响已经导出到 MATLAB 命令行空间中的数据，即使关闭了“Network/Data Manager”窗口，命令行空间中的数据也依然存在。

第8章　Simulink

Simulink 工具箱包含大量的动态仿真库，能够对实际系统进行动态仿真，而且可以非常方便地实现 Simulink 与 MATLAB 之间的交互操作，用 MATLAB 命令进行 Simulink 模型的仿真、数据交换等，同时可以编写 M 函数或者 S 函数进行复杂系统的 Simulink 仿真。在此主要介绍 Simulink 建模与仿真的基本知识以及应用实例分析。

8.1　Simulink 概述

Simulink 是一个用来进行动态系统建模、仿真和分析的软件包。Simulink 提供了大量的仿真元件库，可以非常方便地搭建、分析和仿真各种动态系统，包括连续系统、离散系统和混合系统。Simulink 交互式的开发界面方便用户直接拖放、连接元件，设置模块属性和仿真参数，实现动态系统的建模仿真。

同时，Simulink 强大的扩展功能，包括 M 文件编程技术、S 函数编程级数、子系统创建封装技术等方便用户开发自己的仿真工具箱；Simulink 中集成了大量的专业模块库，包括信号处理 DSP 模块库、电力系统仿真模块库、通信系统模块库等，广泛应用于神经各领域中。综合起来，Simulink 工具箱具备以下强大的功能。

1）优越的交互式界面，操作简单

Simulink 模块库浏览窗口提供了大量的仿真元件库以及其他专业模块库，用户只需启动 Simulink 仿真环境，在 Simulink Library Browser 的窗口选择需要的模块，拖放元件，然后连接，就可以建立仿真模型，同时子系统概念可以帮助用户对独立的仿真功能模块进行封装，使系统模型更加简洁、清晰。这种简单的元件拖放、连接、仿真的过程，使 Simulink 建模变得非常容易直观。

2）丰富的模块库，应用领域广泛

Simulink 仿真环境下提供了大量的模块库，包括连续模块库、离散模块库、非线性模块库、信号与系统模块库、数学模块库、输入/输出模块库及子系统模块库等。通过使用这些模块库，能够非常轻松地描述实际物理模型。

3）模块库易于扩展、移植性好

Simulink 仿真平台中集成了大量的模块库，能够满足一般用户的仿真需求。同时 Simulink 子系统的概念可以方便地实现模块库扩展。在一些大型系统仿真中，仿真模型元件众多，此时需要建立各种独立功能模块子系统，并进行封装，使之成为一个独立的功能子模块，用户可以将自己建立的子系统组成模块库，并添加到 Simulink 仿真平台的库浏览窗口中，方便使用。

对于一些复杂系统，Simulink 仿真环境下可以使用 S 函数来实现，它可以使用 MATLAB、C、C++、Fortran 和 Ada 等语言来编写，实现连续系统、离散系统和混合系统等，这些不同语言下的相关数值算法，不用修改就可以直接移植到 Simulink 开发环境下，因此模块库的移植性非常好。

4）仿真手段灵活，接口丰富

在 Simulink 仿真环境下，仿真手段灵活，一方面可以通过交互式界面菜单按钮的使用进行系统模型仿真，同时也可以通过命令形式进行模型仿真。

交互式界面的仿真方法简单快捷，但是当需要对仿真模型进行重复仿真，或者在仿真模型中模块参数动态变化等情况下，这种交互式操作的仿真手段会带来极大不便。

于是命令行仿真手段凸显优势，通过命令行仿真，可以实现动态改变模型参数，进行多次模型的重复仿真和数据分析等。在 Simulink 仿真环境下，与 MATLAB 数据交换的方式很灵活，非常方便用户对仿真结果进行数据整理分析。

8.2 Simulink 启动以及建立文件

在了解了 Simulink 基本组成和功能介绍后，可以利用 Simulink 仿真平台建立仿真模型，对实际物理模型进行仿真。本节将主要介绍 Simulink 仿真平台的启动、仿真模型的建立以及 Simulink 库文件的建立。

8.2.1 Simulink 启动

在启动 Simulink 仿真环境之前，应该首先启动 MATLAB。在 MATLAB 工作环境中，启动 Simulink 主要有以下 3 种方法。

（1）用命令行 Simulink 启动 Simulink 开发环境，在 MATLAB 命令窗口（Command Window）中直接输入命令

```
>> Simulink
```

按“Enter”键后就会新开启一个“Simulink Library Browser”的窗口，如图 8-1 所示。

（2）第二种方法是使用 MATLAB 工具栏按钮“ ”启动 Simulink 仿真平台，如图 8-2 所示。

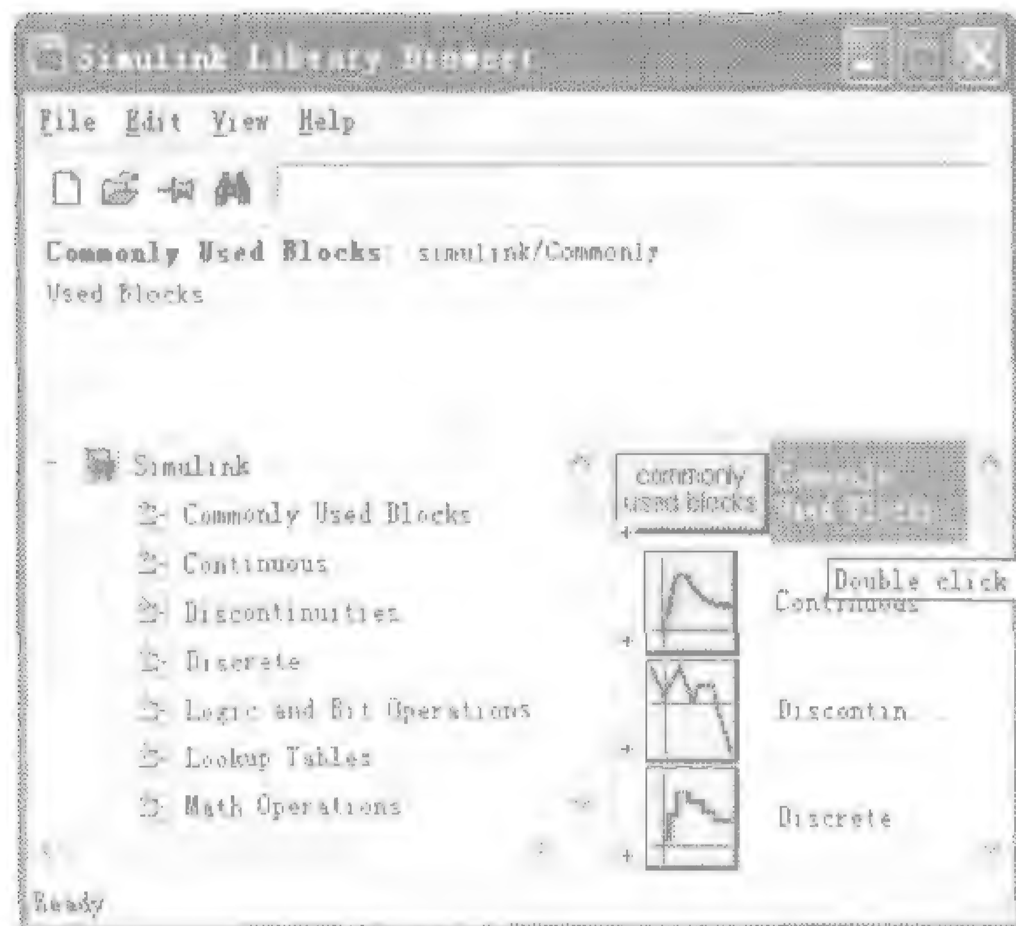


图 8-1 “Simulink Library Browser”窗口

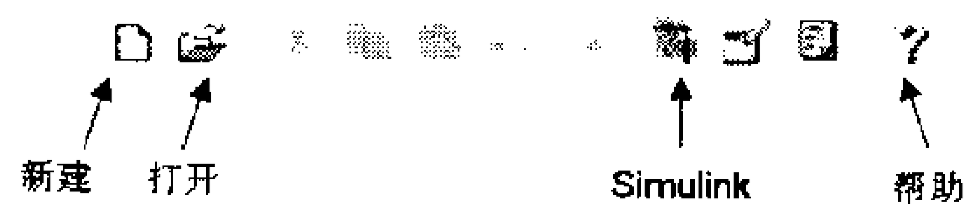


图 8-2 Simulink 启动按钮

（3）用命令行 Simulink3 启动 Simulink 开发环境，在 MATLAB 命令窗口（Command Window）中直接输入命令

```
>> Simulink3
```

按“Enter”键后就会启动一个标题为“Library：simulink3”的新窗口，如图 8-3 所示。从图中可以看到 Simulink 库中的一些主要模块库。双击模块库，即可看到对应模块库中的元件列表。例如双击连续模块库（Continuous），或者单击鼠标右键后选择“Open Block”选项，就可以在新窗口中看到连续模块库中所包含的模块元件，如图 8-4 所示。

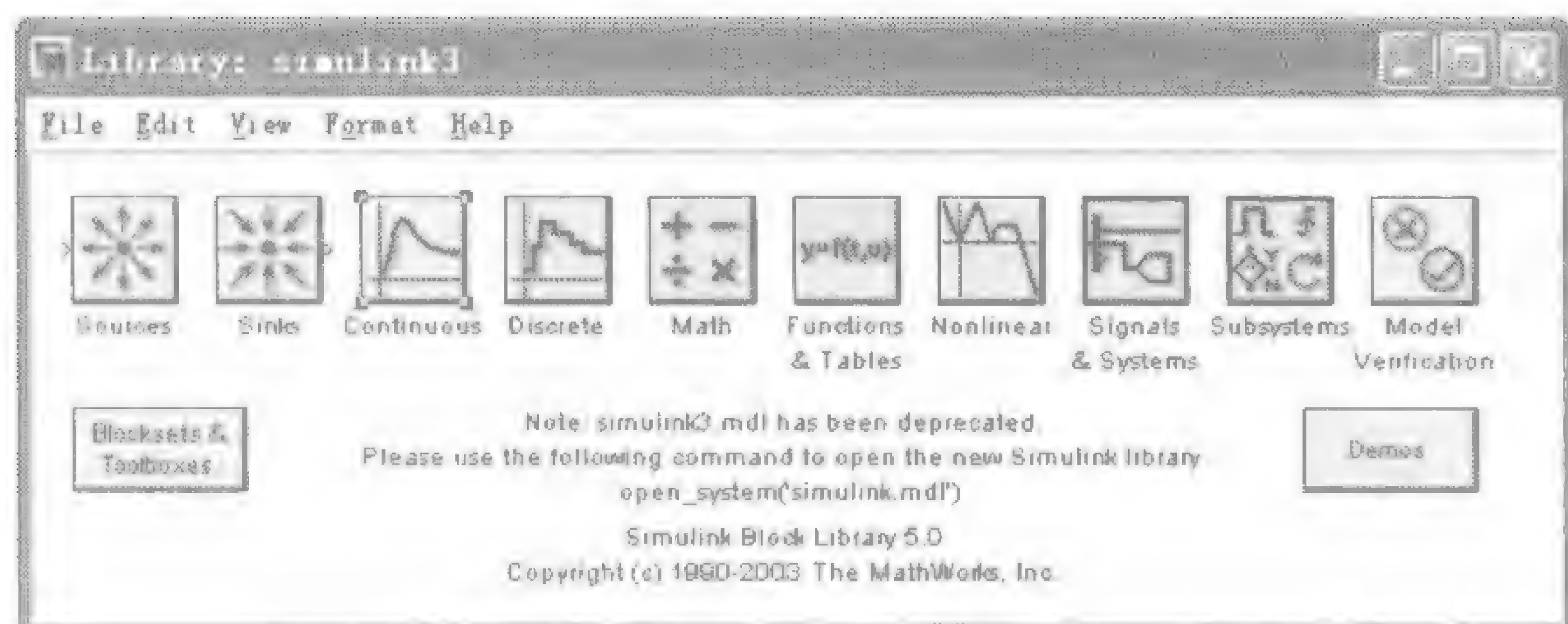


图 8-3　用 Simulink3 启动 Simulink

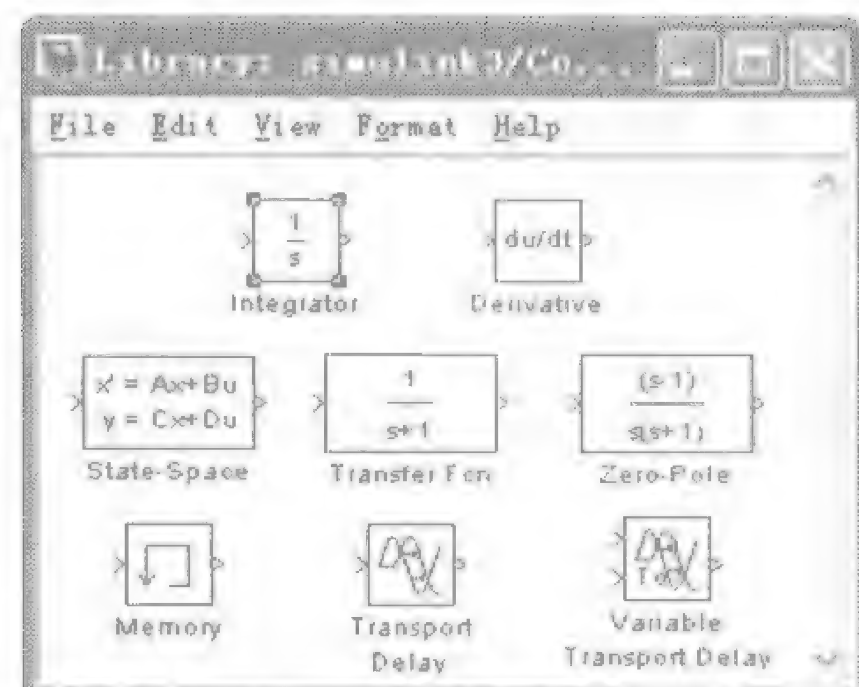


图 8-4　连续模块库中的元件列表

8.2.2　MDL 文件的建立

启动了 MATLAB/Simulink 仿真平台后，需要建立一个空白的仿真模型，即.mdl 文件。新建一个空白的系统仿真模型可以有以下几种方法。

（1）单击主窗口“File”菜单下的“New”子菜单，选择“Model”选项，或者在 Simulink 主窗口上按“Ctrl+N”快捷键同样可以创立空白的仿真模型。

（2）单击 Simulink 主窗口中工具栏中的“🗋”建立一个空白的仿真文档，用“📂”打开一个当前路径下保存过的仿真模型。

（3）在 MATLAB 主窗口的“File”菜单下的“New”子菜单中，选择“Model”选项。

如图 8-5 所示为新建的仿真模型，按“Ctrl+S”快捷键或者选择菜单栏“File”菜单下的“Save”选项，输入仿真模型的文件名，然后设置模块参数与系统的仿真参数，就可以完成一个简单系统的仿真例子。在这个仿真模型中，使用了输入源模块库（Sources）中的正弦波发生器模块（Sine Wave）、数学模块库（Math Operations）中的绝对值模块（Abs）以及输出模块库（Sink）中的示波器模块（Scope）。3 个模块的功能叙述以及参数设置如下。

① 正弦波发生器模块（Sine Wave）：它用来产生一定幅值和一定频率的正弦波信号 $x(t)=(A+\Delta A)\sin(wt+\varphi_0)$。同时可以设置频率采样或者时间采样、正弦波信号的初始相位和偏移量。右键单击该模块，即可对模块进行参数设置。在“Sine type”的下拉菜单下可以选择基于时间或者基于频率的正弦波信号类型。Amplitude、Bias 可以分别设置幅值和偏移量，对应于 A 和 ΔA，Frequency 和 Phase 下分别设置正弦波信号的频率和初始相位，注意单位是 rad/sec 和 rad。最后设置采用时间“Sample time”，如图 8-6 所示。

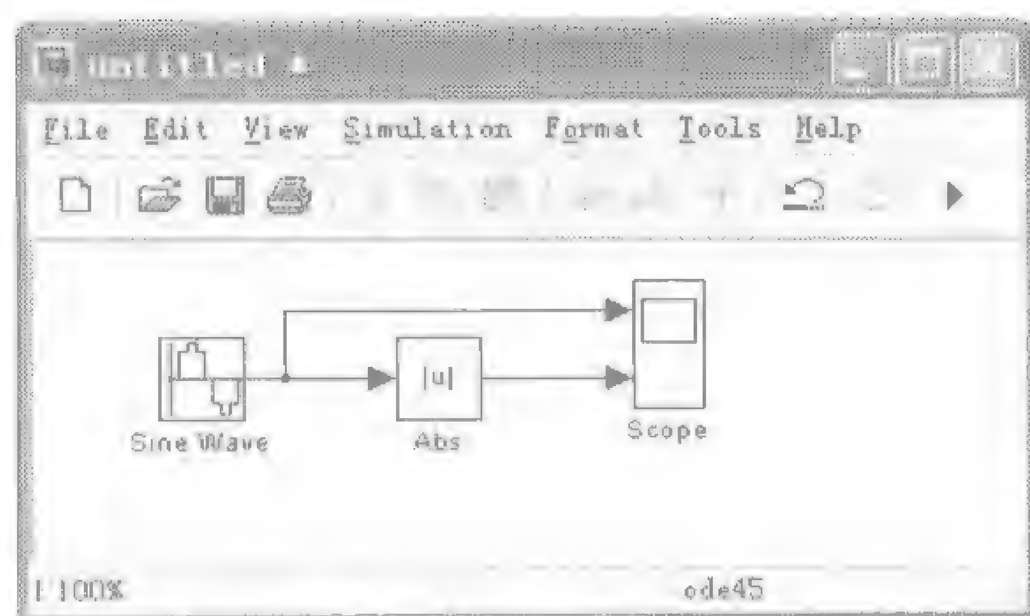


图 8-5　新建的仿真模型



图 8-6　正弦波发生器模块属性设置

② 取绝对值模块（Abs）：实现绝对值功能，属性设置使用默认设置。

③ 示波器模块（Scope）：示波器默认的情况下只有一个输入端，但很多情况下为了观测多

路不同的输出信号，需要增加示波器输入端口，这里需要设置示波器的属性。双击 Scope 模块，单击工具栏按钮"[icon]"，弹出如图 8-7 所示的示波器参数设置窗口。



（a）"General"属性页　　（b）"Data history"属性页

图 8-7　示波器模块属性设置

在"General"属性页中，根据需要的输入端口数量设置"Number of axes"。"Time range"编辑框中输入示波器时间轴的显示范围。在"Data history"属性页中，去掉"Limit data points to last"选项，使信号能够完全显示于示波器中。如果需要对示波器的数据进行处理，可以选择"Save data to workspace"选项，定义保存变量的名称及数据类型。

在保存示波器数据时，提供 3 种不同的数据格式：第一种是"Structure with time"，第二种是"Struct"，第三种是"Array"。

（4）当完成了仿真模型所有模块的参数设置并且正确地设置系统仿真参数后，就可以进行模型的仿真。在模型窗口（如图 8-5 所示）中，单击菜单栏"Simulation"下的"Configuration parameters"选项，或者按"Ctrl+E"快捷键，即可弹出仿真参数设置对话框，如图 8-8 所示。

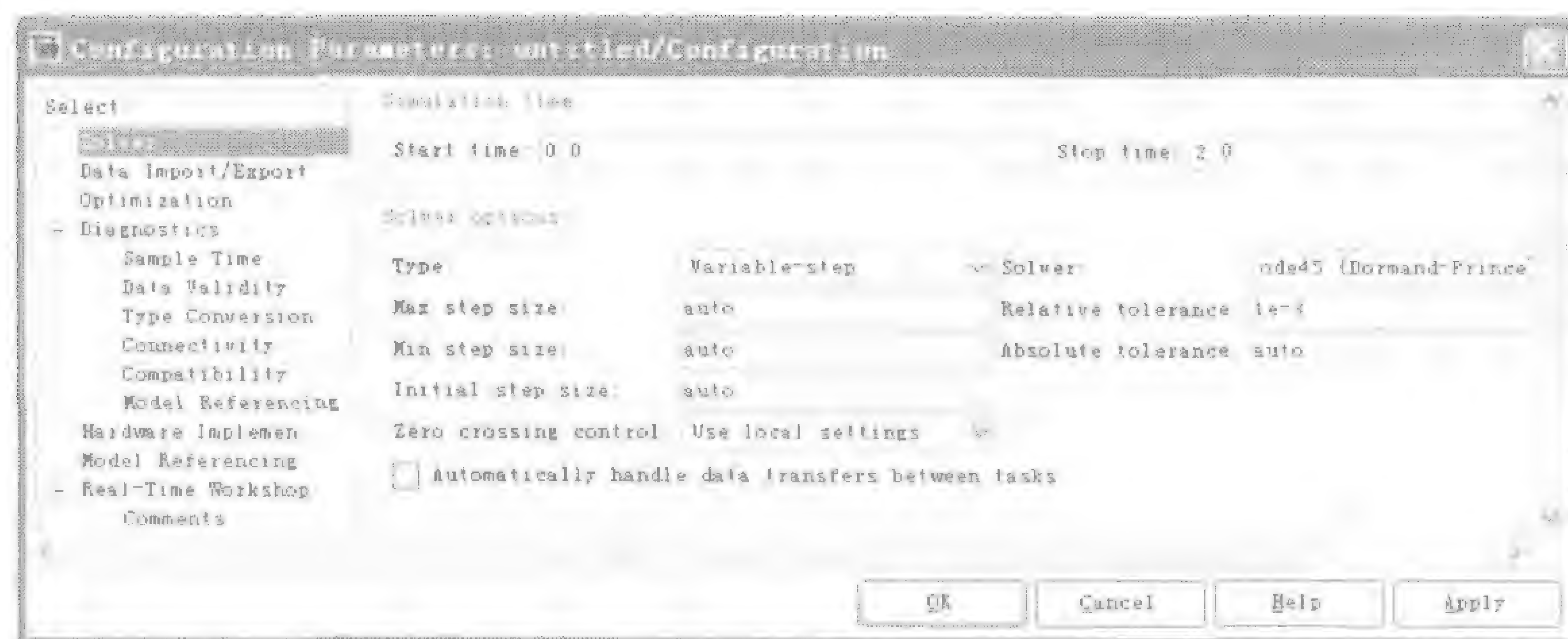


图 8-8　仿真参数设置对话框

当仿真模型所有模块参数和仿真参数设置完毕后，就可以进行系统的仿真了。单击模型窗口（如图 8-5 所示）中菜单栏"Simulation"下的"Start"选项，或者按快捷键"Ctrl+T"或者单击工具栏按钮"▶"进行模型仿真，结果如图 8-9 所示。

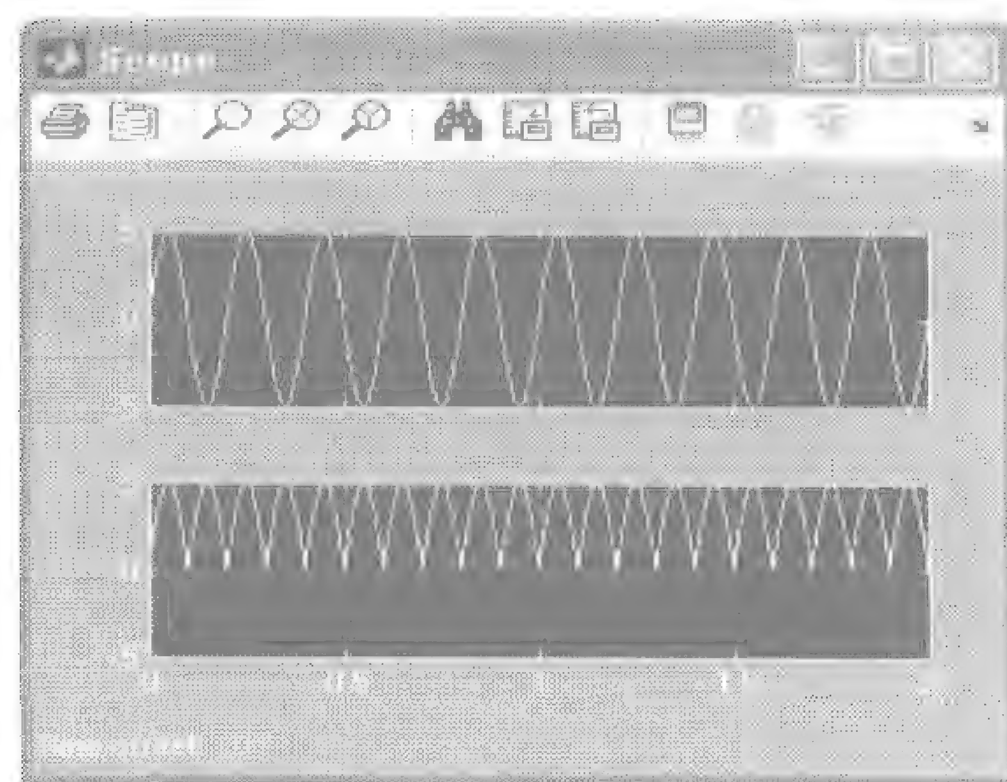


图 8-9　仿真结果

示波器在保存的不同数据格式下，数据结构不尽相同。当数据格式设置为"Structure with time"时，仿真结束后，在命令窗口中输入

```
>> compare_sine
compare_sine =
        time: [200001×1 double]
```

```
    signals: [1×2 struct]
  blockName: 'untitled/Scope'
```

可以看出，示波器保存的是包含时间和信号数据的结构体，同时还有示波器模块的名称。而信号数据也是一个结构体。在命令窗口中输入

```
signals_struct=compare_sine.signals
signals_struct =
1×2 struct array with fields:
    values
    dimensions
    label
    title
    plotStyle
```

如果要获取时间和信号的数据可以分别用以下命令

```
>> time=compare_sine.time;   %获取仿真时间数据
>> pre_signal=compare_sine.signals(1).values;   %获取第一个信号的数据
>> aft_signal=compare_sine.signals(2).values;   %获取第二个信号的数据
```

当数据格式设置为 Struct 时，仿真结束后，在命令窗口中输入

```
>> compare_sine
compare_sine =
         time: []
      signals: [1×2 struct]
    blockName: 'compare_sine/Scope'
```

发现此时示波器数据中时间数据为空集，并且没有保存时间的数据。信号数据为 1×2 的结构体数据。

第三种数据结构 Array 必须是示波器为单输入时才能保存为数组。在多输入情况下，必须以结构体形式保存数据。

示波器图形编辑和数据处理方法。

从图 8-9 可以看出，示波器仿真结果无法直接进行编辑，而且以黑色为图形背景，不利于工程论文图形美观需求。因此需要对示波器图形或者数据进行重新处理。以下提供 5 种基本的处理方式。

① 按下键盘的屏幕打印按钮“PrintScreen”，然后粘贴到“Windows”位图编辑器，即 Windows 下自带的画图板软件，选择图像单击右键，选择“反色”即可。

② 在“Scope”属性页中的“Data history”属性页，选中“Save data to workspace”单选框，然后在“Variable Name”编辑框中指定变量名，“Format”下拉菜单下选择示波器保存数据类型，将仿真数据保存在工作窗口下，使用 plot 命令绘制仿真结果，如图 8-10 所示。

③ 直接在模型的输出信号线上添加“To Workplace”模块，在工作窗口中将仿真数据使用 plot 命令绘制出来。

④ 直接在模型的输出信号线上添加 Outport 模块，用 plot 命令绘制 tout 和 yout。

⑤ 模型仿真结束后，等示波器 Scope 显示出图像以后，在 MATLAB 的命令行窗口中输入

```
>> set(0,'ShowHiddenHandles','on')
>> set(gcf,'menubar','figure')
```

单击菜单栏的“nsert”选项，光标会变成“十”字形状，然后在图像的任意一处双击左键出现一个“proprtyEditor”对话框，选中“style”，在窗口的右边会出现“color”选项，这时就可以任意修改波形背景颜色，同时也可以对曲线颜色及坐标轴进行设置，结果如图 8-11 所示。

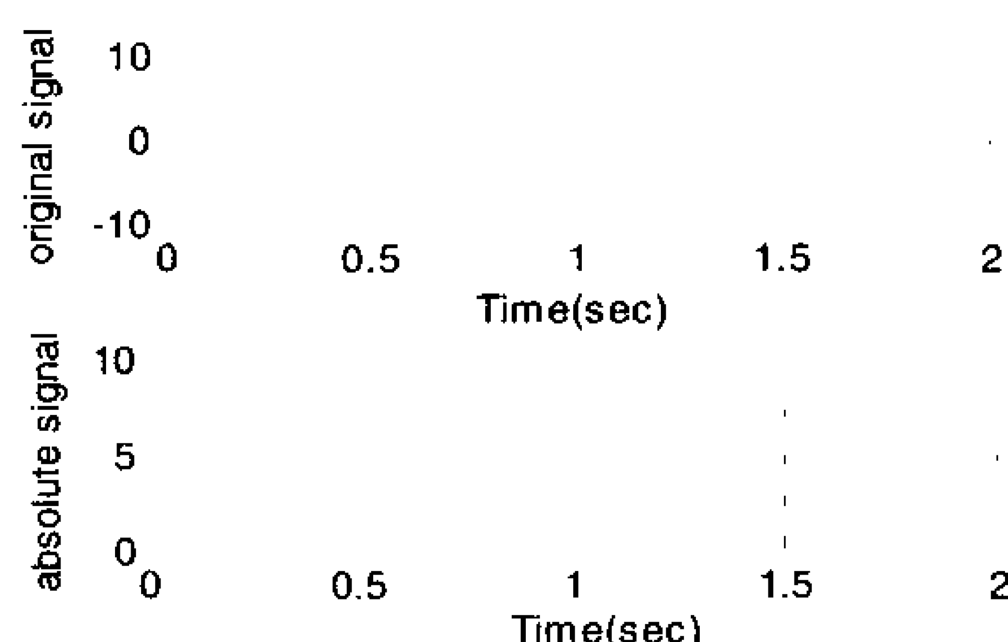


图 8-10　示波器数据处理图形

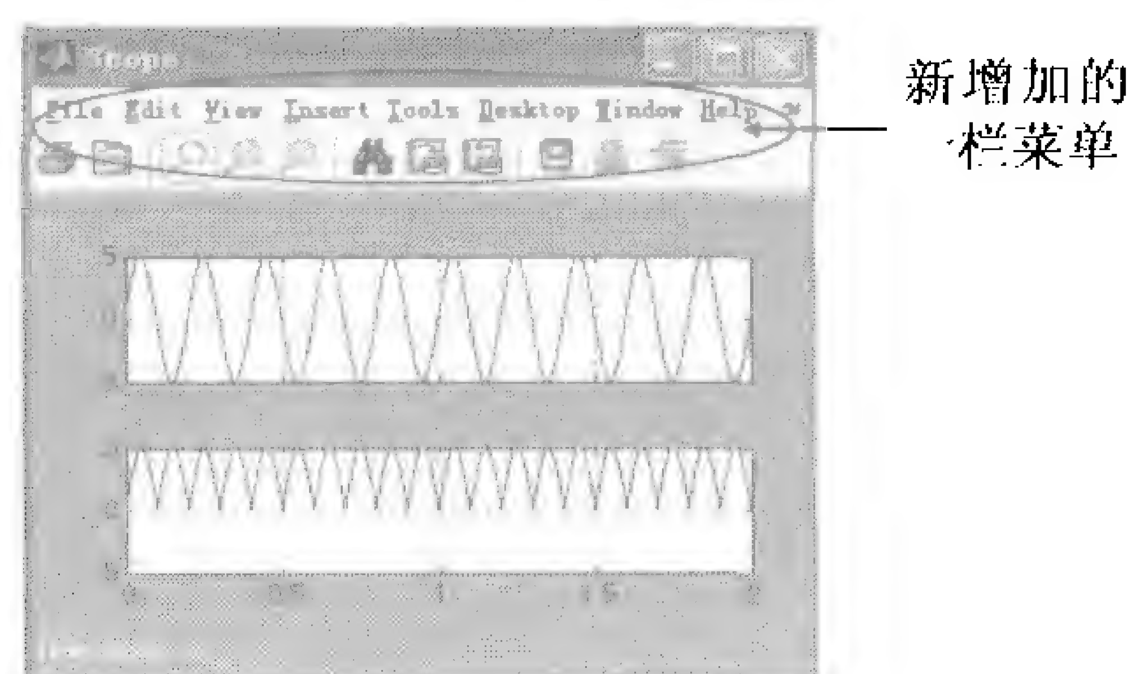


图 8-11　修改结果

在命令行窗口中输入以下绘图命令，结果如图 8-10 所示。

```
subplot(211)
time=compare_sine.time;
pre_signal=compare_sine.signals(1).values;
aft_signal=compare_sine.signals(2).values;
plot(time,pre_signal)
grid on
xlabel('Time(sec)');
ylabel('original signal')
subplot(212);
plot(time,aft_signal)
grid on;
xlabel('Time(sec)');
ylabel('absolute signal');
```

以上通过一个简单的系统模型演示了 Simulink 建模与仿真的基本步骤。读者通过这一部分的学习能够大致了解一个 Simulink 仿真模型建立的步骤、示波器数据图像化显示的基本方法以及仿真数据的交互式处理。在后续章节中，会更加详细地介绍 Simulink 模块库各模块的功能、使用方法和应用场所，通过更加深入的学习，读者能够轻松地建立相对简单的 Simulink 仿真模型。对于复杂系统、离散一连续混合系统的仿真，在介绍完命令行仿真技术后希望读者能够进一步对这些内容进行学习。

8.2.3　Simulink 库文件的建立

Simulink 仿真平台可以建立用户自定义的库文件，并将它们显示在“Library Browser”窗口下，方便用户进行模块的操作。用户可以将一些平时使用比较频繁或者自己建立的一些封装子模块集中在一起，方便使用。要建立 Simulink 库文件，首先启动 Simulink 工作窗口，单击菜单“File”下的“New”子菜单，选择“Library”选项，打开一个新的 Library 窗口界面，此时，用户可以将自己需要添加的一些模块加入到新的窗口中，如图 8-12 所示。然后保存为所需要定义的库文件名称。虽然按照上述方法，可以将一些用户常用的模块集中在一起方便使用，但是，每次使用都需要打开自定义模块窗口，而无法像 Simulink 仿真模块一样显示在“Library Browser”窗口下。按照以下步骤可以改变这种情况。

（1）首先建立一个库文件，将自定义的封装模块加入库文件中，如图 8-12 所示，保存文件，在这里，命名为 own_function.lib。

（2）在 MATLAB 路径下创建存放库文件的路径，也即自定义的模块库在 Library Browser 下的显示位置。注意在 MATLAB 中，每一个模块库文件所在的路径必须不相同。创建的 MATLAB 路径是 C:\Program Files\MATLAB71\work\myLibrary（根据自己的 MATLAB 安装路径来确定），其中 myLibrary 为自定义的文件夹，如图 8-13 所示。

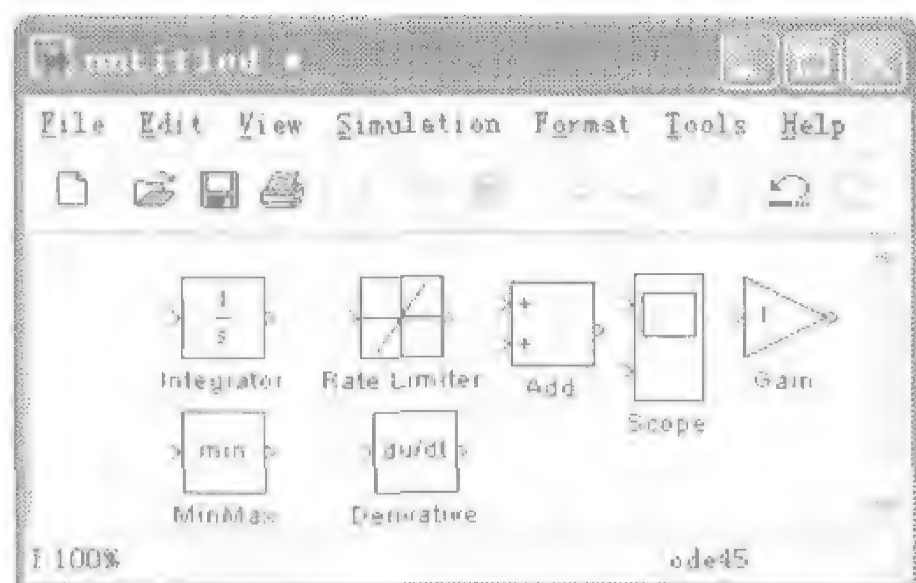


图 8-12 库文件窗口

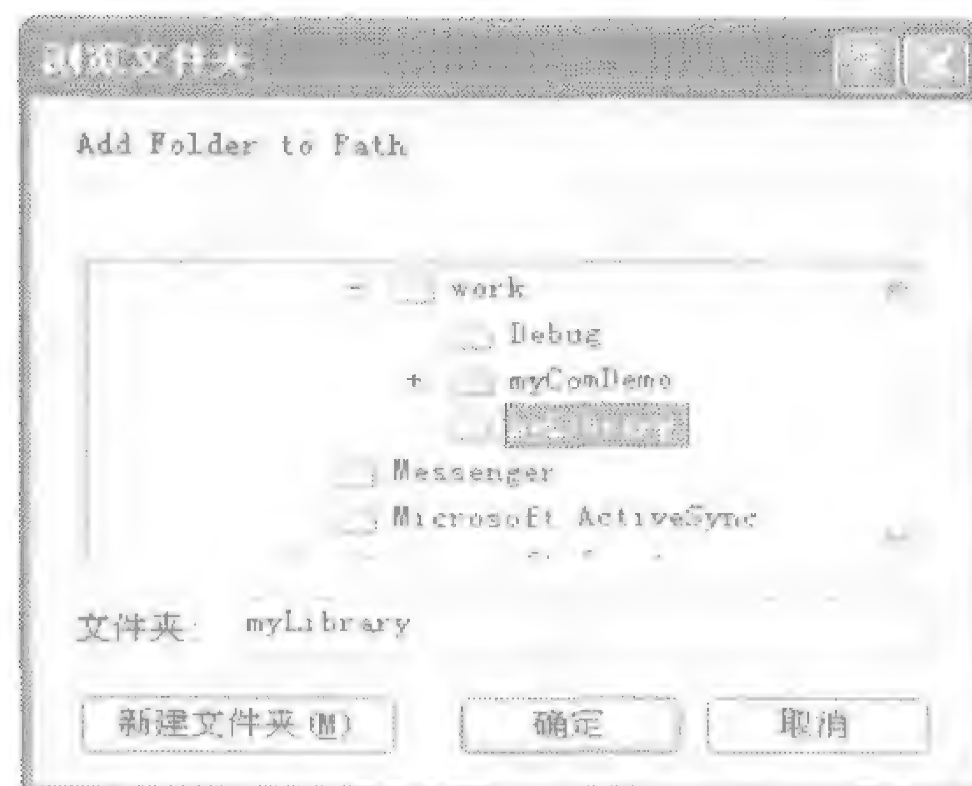


图 8-13 添加模块路径

（3）将第一步中建立的库文件复制到新建的 MATLAB 路径下。在 MATLAB 主窗口的“File”菜单栏下选择“Set Path”选项，然后单击“Add Folder”按钮，将新建的路径添加进来，如图 8-14 所示，然后保存（Save）、退出（Close）。

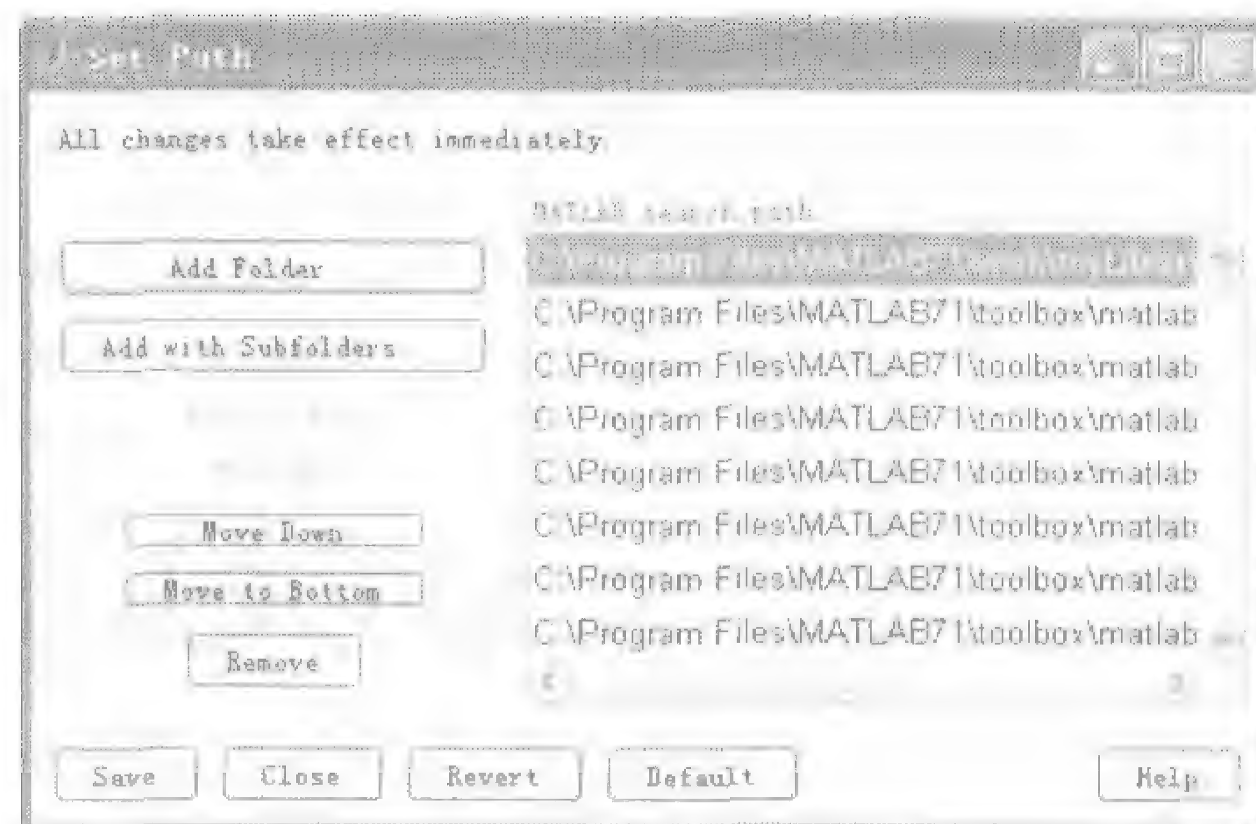


图 8-14 已添加自定义模块路径的显示窗口

（4）显示自定义的模块库，还需要复制 slblocks.m 函数到新建的路径下。在 MATLAB 的命令窗口输入

```
which('slblocks.m','-all')
open('C:\Program Files\MATLAB71\toolbox\simulink\blocks\slblocks.m')
```

这样就可以打开 slbocks.m 文件模板，为了将自定义的模块库显示在“Library Browser”窗口下，需要对该程序进行修改。在本演示中，模块库名称为 own_function.lib，因此对应的 slbocks.m 文件为

```
function blkStruct = slblocks
%SLBLOCKS Defines the block library.
%Library's name. The name appears in the Library Browser's
%contents pane.
%模块库的显示名称
blkStruct.Name = ['own Definition' sprintf('\n') 'Library'];
% The function that will be called when the user double-clicks on
% the library's name.
%自定义的模块库名称
blkStruct.OpenFcn = 'own_function';
% The argument to be set as the Mask Display for the subsystem.   You
% may comment this line out if no specific mask is desired.
% Example:   blkStruct.MaskDisplay = 'plot([0:2*pi],sin([0:2*pi]));';
```

```
% No display for Simulink Extras.
%
blkStruct.MaskDisplay = '';
% End of slblocks
```

按照以上的代码修改文件后，将其复制到自定义的路径下，这样在原来自定义的路径下就包含了两个文件，一个是自定义模块库文件，另一个则是 slblokcs.m 文件。

（5）重新启动 MATLAB 环境，启动 Simulink 仿真平台，如图 8-15 所示，可以发现在“Simulink Library Browser”窗口下增加了一个新的模块库，名称是 own Definition Library，展开后，可以看到内部的自定义模块。那么用户可以方便地进行拖放模块，可以使用自定义的封装模块了。

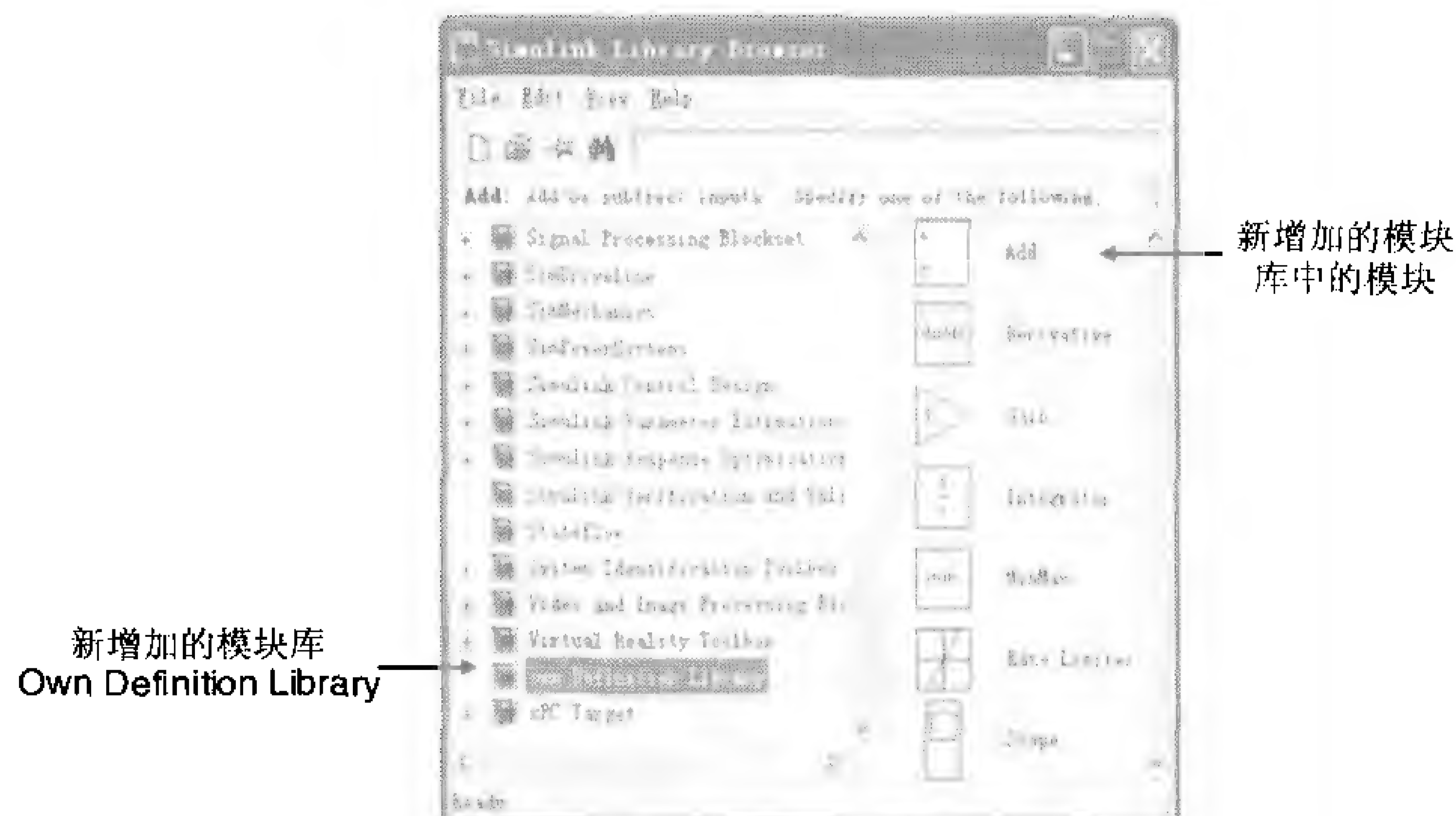


图 8-15　“Simulink Library Browser”窗口下显示自定义模块库

8.3　Simulink 交互式仿真集成环境

8.3.1　Simulink 仿真

Simulink 模型本质上是一个计算机程序，它定义了描写被仿真系统的一组微分或差分方程。当选中模型窗口菜单“Simulation”→“Start”时，Simulink 就开始用一种数值解算方法求解方程。

Simulink 仿真参数都有内定的默认值，也可以对各种仿真参数进行重新配置，包括仿真的起始和终止时刻的设定、仿真步长的选择、各种仿真精度的选定、数值积分算法的选择、是否从外界获得数据、是否向外界输出数据等。

选中模型窗口菜单“Simulation”→“Configuration Parameters”，就可以出现仿真参数配置对话框。本节简单介绍解算器参数设置页和仿真数据的输入/输出设置页。

解算器参数设置在“Configuration Parameters：untitled/Configuration”窗口中“Solver”菜单下，如图 8-8 所示。

（1）仿真时间（Simulation time）区域。

① Start time：仿真起始时间。

② Stop time：仿真结束时间。

（2）解算器选项（Solver options）区域。

① Type 和 Solver：Type 用来选择解算器的类别（变步长和定步长），Solver 用来选择解算器的具体算法类型（ode45、ode23、ode113、ode15s 等）。

② Max step size：最大步长。

③ Min step size：最小步长。

④ Initial step size：初始步长。

⑤ Relative tolerance：相对精度。

⑥ Absolute tolerance：绝对精度。

⑦ Zero crossing control：零穿越控制。

（3）诊断控制（Solver diagnostic controls）区域。

① Number of consecutive min step size violations allowed：达到最小步长的允许次数。

② Consecutive zero crossings relative tolerance：零交叉相对精度。

③ Number of consecutive zero crossing allowed：零交叉允许次数。

仿真数据的输入/输出设置页在“Simulation Configuration Parameters：untitled/ Configuration”窗口中“Data Import/Export”菜单下，如图 8-16 所示。

（1）Load from workspace 区域选择是否从 MATLAB 工作空间导入数据。

① Input：输入数据变量名。

② Initial state：导入数据变量的初始值。

（2）Save to workspace 区域选择是否导出数据到 MATLAB 工作空间。

① Time：选择此项则模型把（时间）独立变量以指定的变量名存储到工作空间。

② States：选择此项则模型把其状态变量以指定的变量名存放于工作空间。

③ Output：选择此项则模型将输出变量以指定的变量名存放于工作空间。

④ Final states：选择此项则模型将最终状态变量以指定的变量名存放于工作空间。

⑤ Signal logging：选择此项则模型将信号已制定的变量名存放于工作空间。

⑥ Inspect signal logs when simulation is paused/stopped：选项此项则在仿真停止或者暂停的时候对信号进行检查。

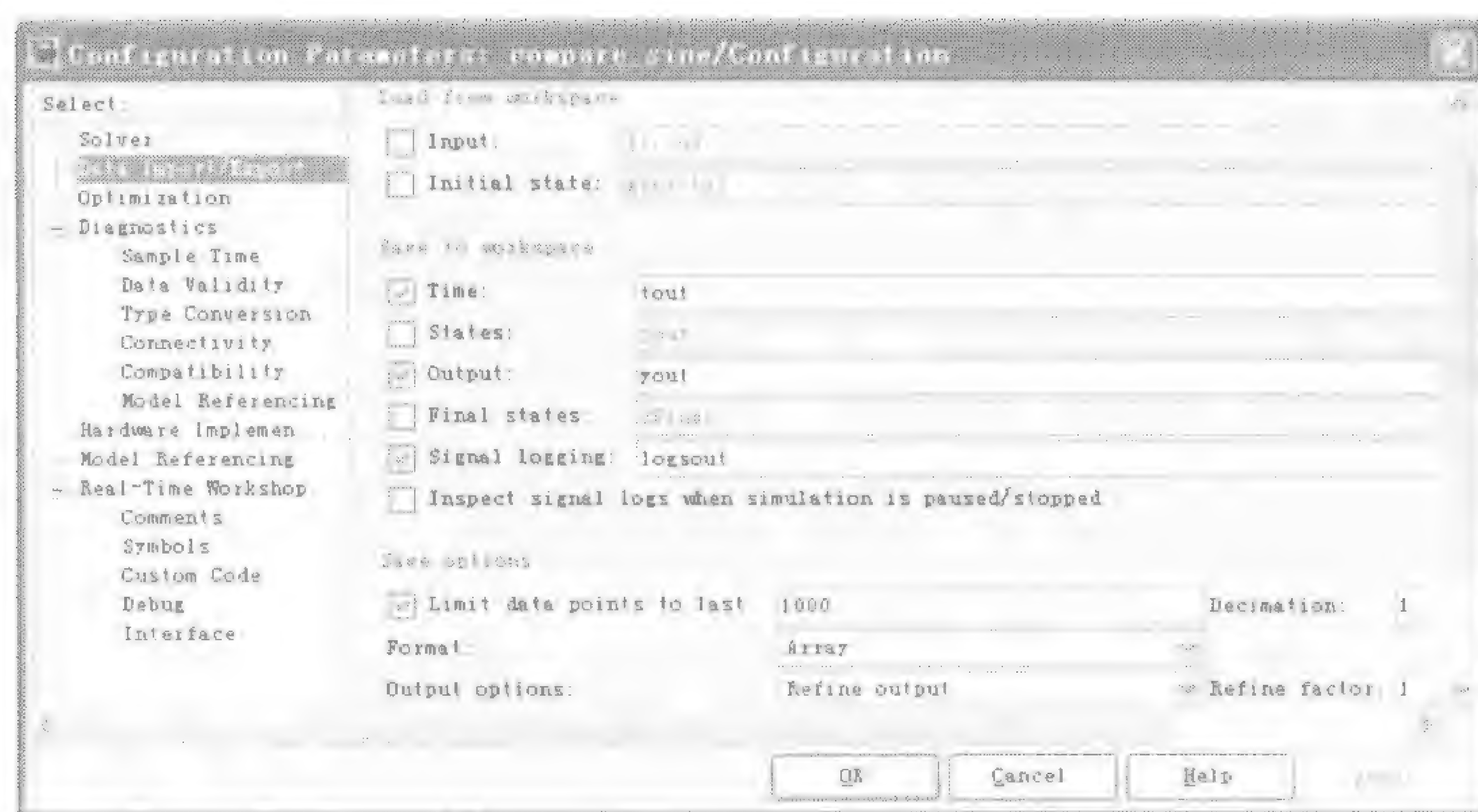


图 8-16　输入/输出参数设置

（3）Save options 区域确定变量存放方式。

① Limit data points to last：选择此项则设定了保存数据的长度。

② Decimation：保存数据的间隔。

③ Format：选择保存数据的格式（数组、结构、带时间的结构）。

④ Output options：输出数据存储模式。

⑤ Refine factor：当选择“Refine output”时，则需要选择这个数据。

8.3.2 Simulink 模型属性设置

在仿真模型的空白处，单击鼠标右键，选择“Model Properties”选项，弹出模型属性设置对话框，如图 8-17 所示。在模型属性设置中，最重要的是“Callbacks”属性页，在这个属性页中，包括了多个不同的回调函数，可以编写程序代码，在 Simulink 仿真时，会在不同时间调用相应的回调函数。

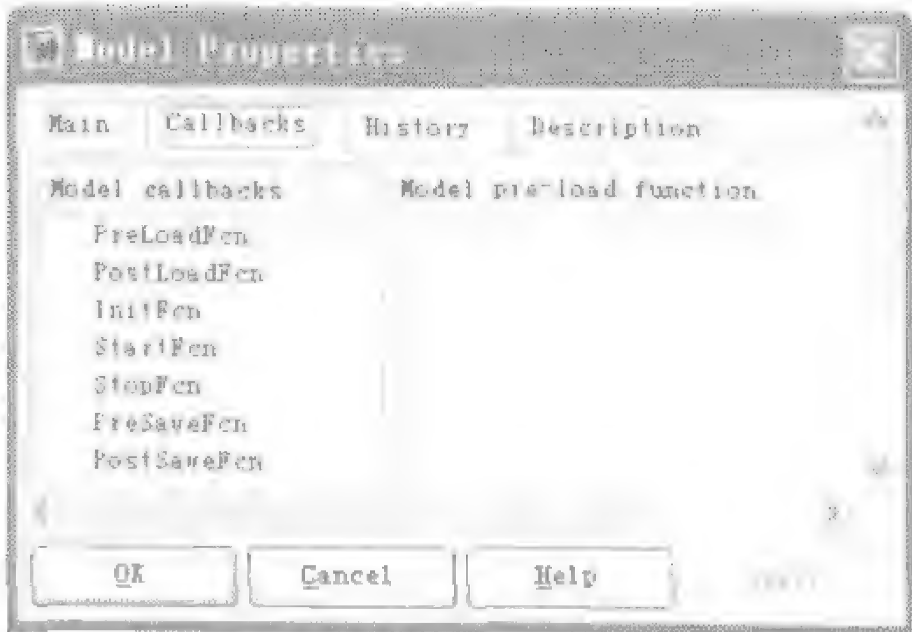


图 8-17　模型属性设置

8.4 Simulink 的模块

MATLAB/Simulink 仿真平台提供了大量、实用的模块库，用户可以使用这些模块库中的元件构建特定的仿真模型。这些模块库主要包括连续系统模块库、离散系统模块库、非线性模块库、信号与系统模块库、数学模块库、接收器模块库、输入源模块库、输出模块库以及子系统模块库等。

用户通过这些基本的模块库可以构建一些复杂的仿真系统，因为 Simulink 仿真库基本上涵盖了所有连续系统、离散系统的基本元件，如果仿真对象能够使用数学模型表示，那么就可以用数学库的相关元件构建子系统。

对于一些动态时变的复杂系统，Simulink 提供了用户自定义函数及 S-function 定义模块的方式。另外，用户也可以将自己建立的子模块集中在自己定义的模块库中，以便能够极大地扩充 Simulink 工具箱。

下面简单地对 Simulink 仿真平台主要的基本模块做一些介绍，在介绍每个基本模块的同时，通过一些应用实例演示相应常用模块的使用。

8.4.1 连续模块库

连续模块库如图 8-18 所示。连续模块库主要包括积分、微分、记忆模块，传输延时模块，状态方程模块，传输函数模块，零极点传输函数模块和变时段的延时模块。各个功能模块的名称和对应的功能描述如表 8-1 所示。

表 8-1　连续模块库中各个模块的名称与功能描述

模块名称	功能描述
Integrator	对连续信号的积分
Derivative	对连续信号的微分
Memory	记忆模块，输出前一时刻系统的输入
Transport Delay	将输入延迟一个给定的时间段
State-Space	实现一个线性系统的状态方程
Transfer Fcn	实现一个线性的传递函数
Zero-Pole	实现一个零极点形式的线性系统
Variable Transport Delay	将输入延迟一个可变的时间段

在连续模块库中，使用较为频繁的是积分、微分、状态方程、传输函数等模块，下面的实例演示这些模块的作用。仿真模型如图 8-19 所示。

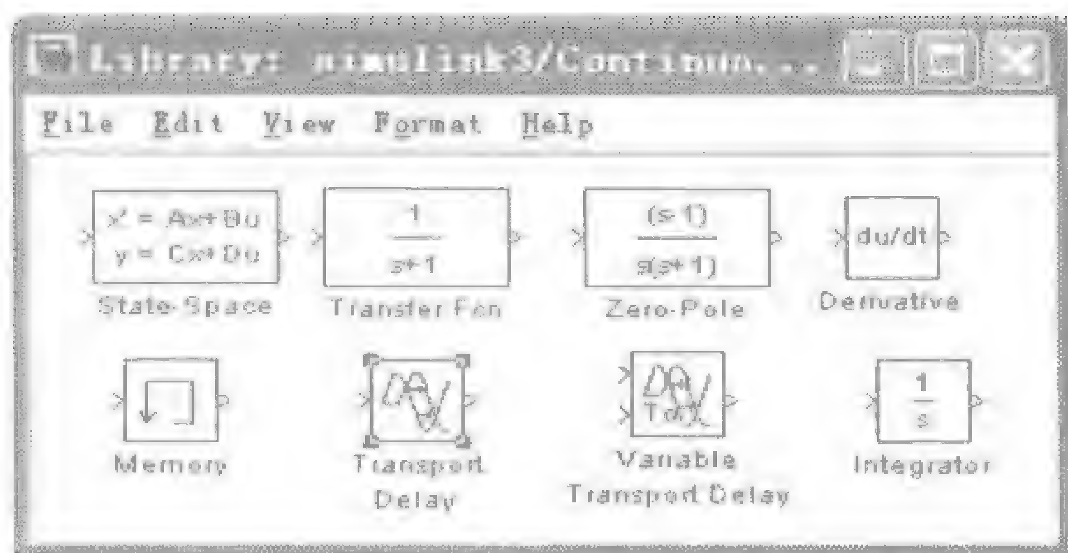


图 8-18　连续模块库

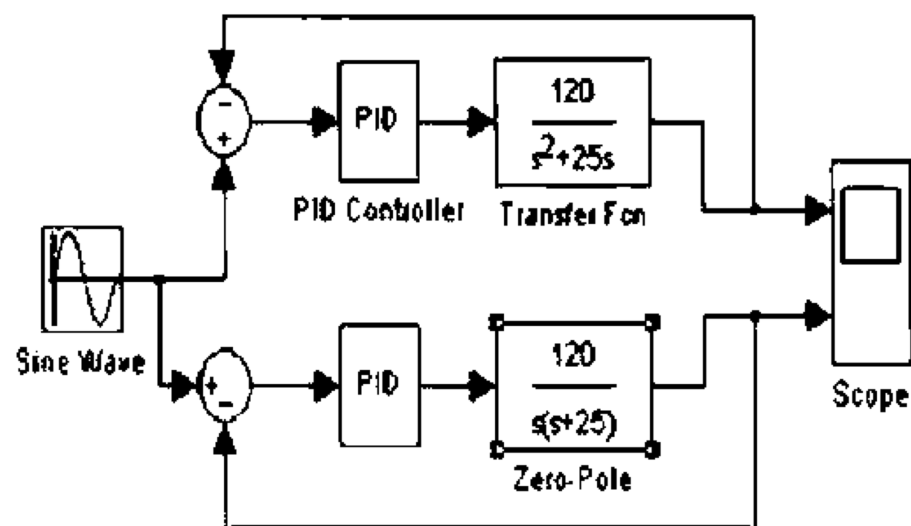


图 8-19　连续库仿真模型实例

在图 8-19 所示的仿真模型中，PID 控制器采用 Simulink 封装形式，实现 PID 调节功能。选中 PID 模块，单击鼠标右键，在弹出菜单选择“look under mask”就可以查看其内部结构，如图 8-20 所示。

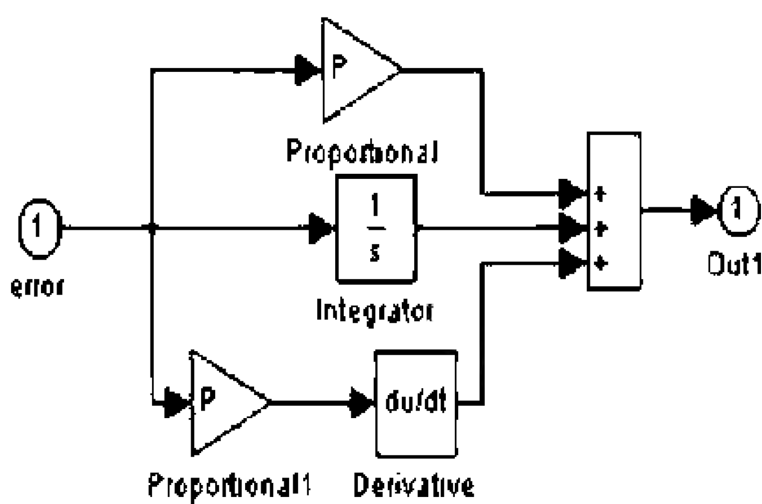


图 8-20　PID 模块封装结构

选择输入源模块库（Source）正弦波模块（Sine Wave）、输出模块库（Sink）的示波器摸块（Scope）和连续模块库（Continuous）的积分模块（Integrator）、微分模块（Derivative）、传输函数模块（Transfer Fcn）、零极点模块（Zero-Pole）和数学模块库（Math）的增益模块（Gain），搭建成如图 8-19 所示的仿真模型，然后进行模块的属性设置。

图 8-21 和图 8-22 所示分别是传递函数模块（Transfer Fcn）和零极点模块（Zero-Pole）的参数设置。当传递函数以分子分母多项式表示时，可以使用 Transfer Fcn 模块，在分子（Numerator）和分母（Denominator）中分别输入传递函数的多项式系统（系数由高次依次下降）。当传递函数以零极点形式表示时，可以使用 Zero-Pole 模块，在零点（Zero）和极点（Pole）处输入传递函数的零极点，同时在增益（Gain）处输入传递函数的增益。在 MATLAB 中，提供了函数进行不同模式之间的转化。

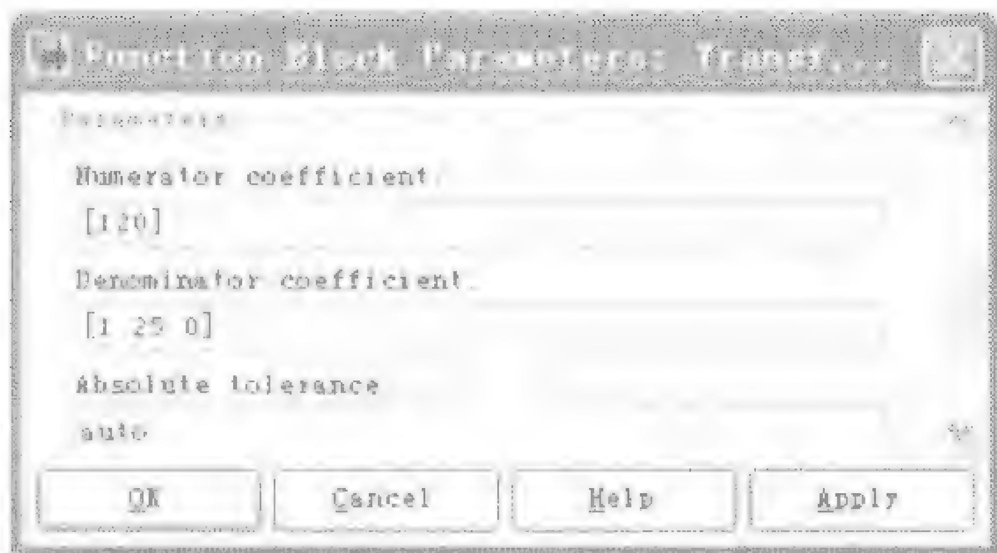


图 8-21　传输函数模块参数设置



图 8-22　零极点模块参数设置

```
[zeros,poles,k]=tf2zp(num,den);   %将传递函数模型转化为零极点模型
[num,den]=zp2tf(zeros,poles,k);   %将零极点模型转化为传递函数模型
```

设置正弦波信号幅值为 5，偏差为 0，频率 5Hz，初始相位为 0。设置 PID 控制器的参数为 Kp=10.75，Ki=1.2，Kd=3.5。采用变步长的 ode23t 算法，仿真时间为 2s，对仿真模型进行仿真。可以看到，两种不同的传递函数表示方法，结果是完全相同的。

8.4.2　离散模块库

离散模块库主要实现离散系统的仿真，包括离散的状态方程、传输函数、采样保持、离散

积分等模块，如图 8-23 所示。表 8-2 所示为离散模块库中各个模块的名称与功能描述。

表 8-2　离散模块库模块名称和功能描述

模块名称	功能描述
Zero-Order Hold	实现一个采样周期的零阶保持器
Unit Delay	延迟输入一个采样周期
Discrete-Time Integrator	对输入信号进行离散的积分
Discrete State-Space	实现一个离散的状态方程
Discrete Filter	实现 IIR 和 FIR 滤波器
Discrete Transfer Fcn	实现一个离散的传输函数
Discrete Zero-Pole	实现一个以零极点表示的传输函数
First-Order Hold	实现一个一阶的采样保持

在离散模块库中，离散积分模块、离散状态空间方程、离散的传递函数和零极点模型与连续库（Continuous）中相应模块的使用完全相同，这里通过仿真模型演示零阶保持器、一阶保持器和单位周期延迟模块的功能差异，仿真模型如图 8-24 所示。

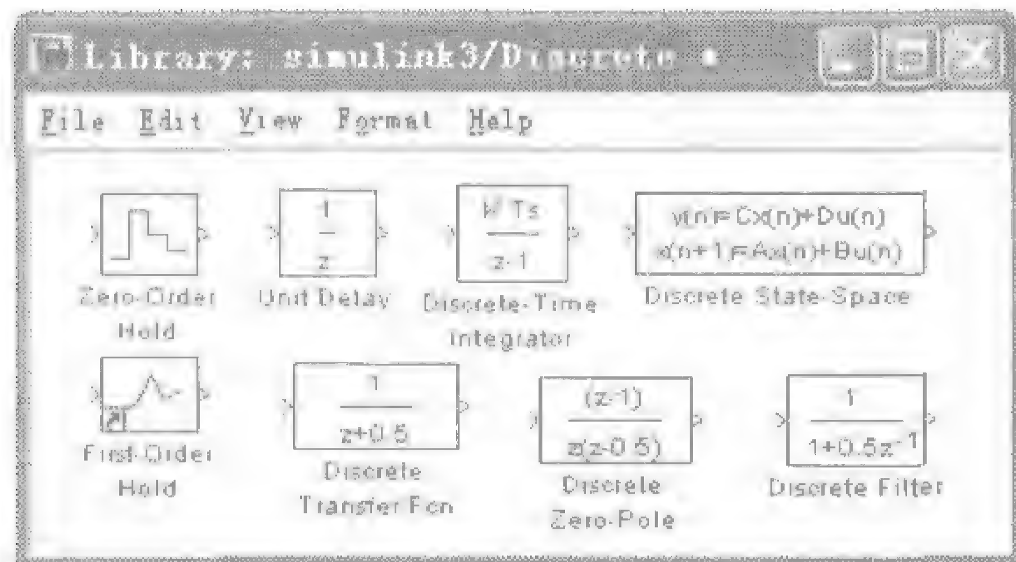


图 8-23　离散模块库

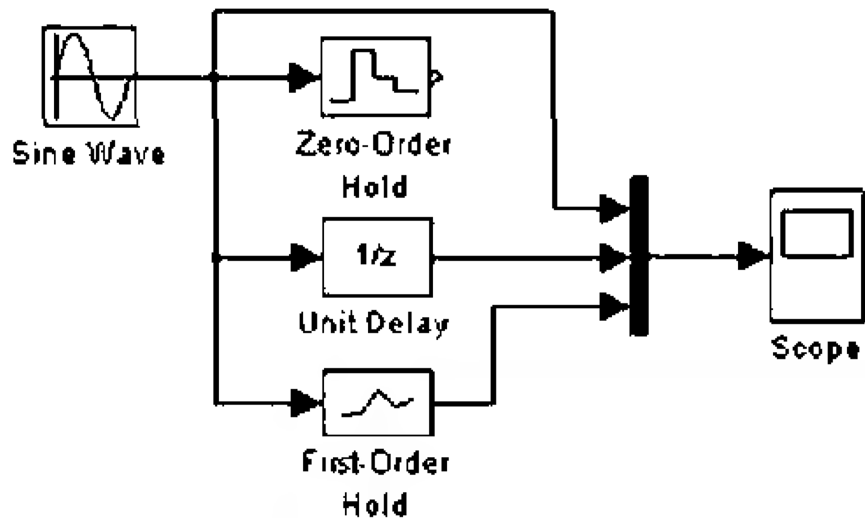


图 8-24　离散模块库演示实例

选择零阶保持器、一阶保持器和单位周期延迟模块，设置它们的采样时间均为 20ms，仿真得到的波形如图 8-25 所示。为了让读者更清楚地观察 3 个模块及其与连续函数的关系，将示波器数据保存，重新绘制，如图 8-26 所示。在 MATLAB 的命令行窗口中输入

```
color_style={'k','r-','b-','m-'};
for i=1:length(color_style)
    plot(ScopeData.time,ScopeData.signals.values(:,i),color_style{i});
    hold on;
end
legend('continuous signal','zero-order hold','unit delay','first-order hold')
```

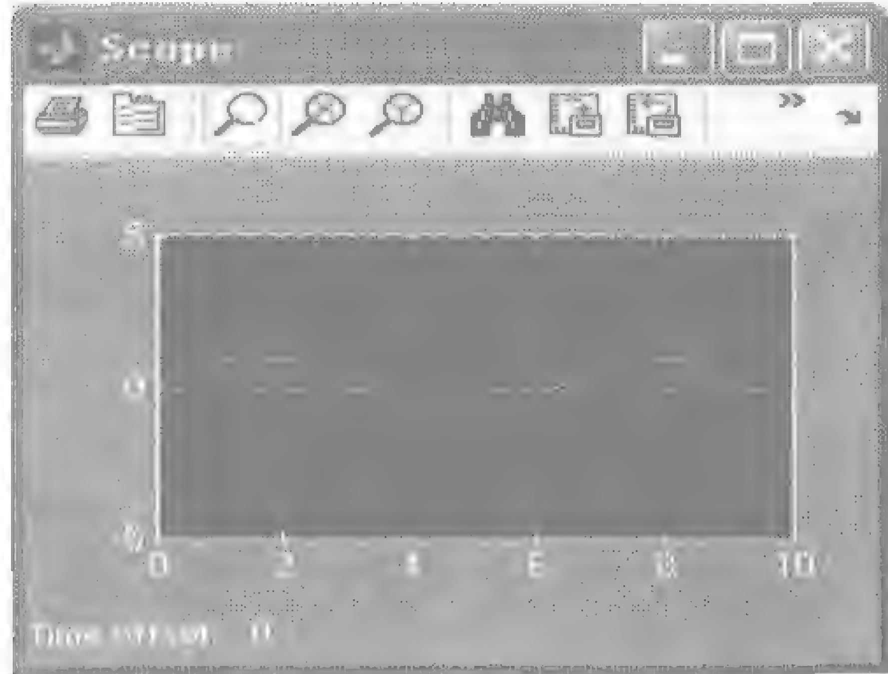


图 8-25　示波器显示结果

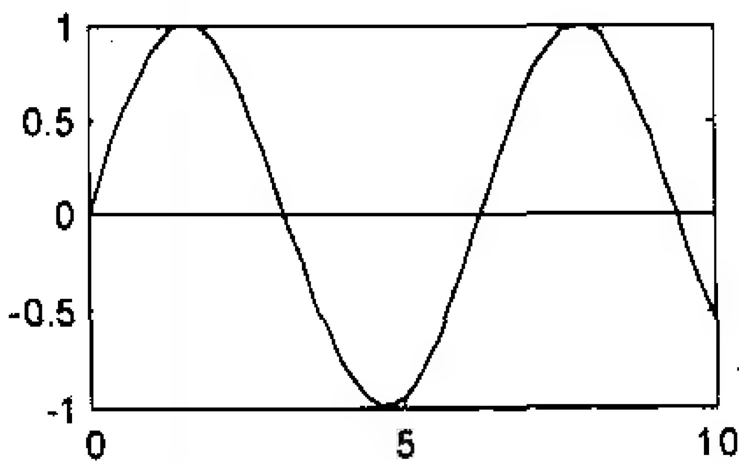


图 8-26　重新绘制后的仿真结果

8.4.3 非线性模块库

非线性模块库主要描述非线性系统的特性，包括死区、离散脉冲、限幅输出和继电器等一些非线性系统，如图 8-27 所示。表 8-3 所示为各个模块的名称和功能描述。

表 8-3 非线性模块库模块名称和功能描述

模块名称	功能描述
Rate Limiter	实现信号变化率的限制
Saturation	输出信号幅值的限制
Quantizer	将输入信号以给定步长离散化
Backlash	将输入信号整体后移一个死区宽度
Dead Zone	在死区区间内将输入信号变为零
Switch	开关，实现信号的开关选择
Manual Switch	手动开关
Relay	继电器
Multiport Switch	多路选择开关
Coulomb & Viscous Friction	输入信号在零点处非连续，在非零点处进行线性放大(缩小)

在非线性模块库（Nonlinear）中，使用较为频繁的是 Switch 模块、Multiport Switch 模块以及继电器模块。图 8-28 所示的仿真模型可以用来选择不同的区间输出，即实现如表 8-4 所示的输出，输入为角度，输出为角度对应的区间号 1~6。仿真模型中，Switch 开关的阈值设定从 Switch～Switch4 依次为 4π/3、π、2π/3、π/3 和 0，MATLAB Fun 调用 mod 函数将输入的角度转化到 0～2π 的区间内。常数信号输入和 MATLAB Fun 的属性设置如图 8-29 和 8-30 所示。

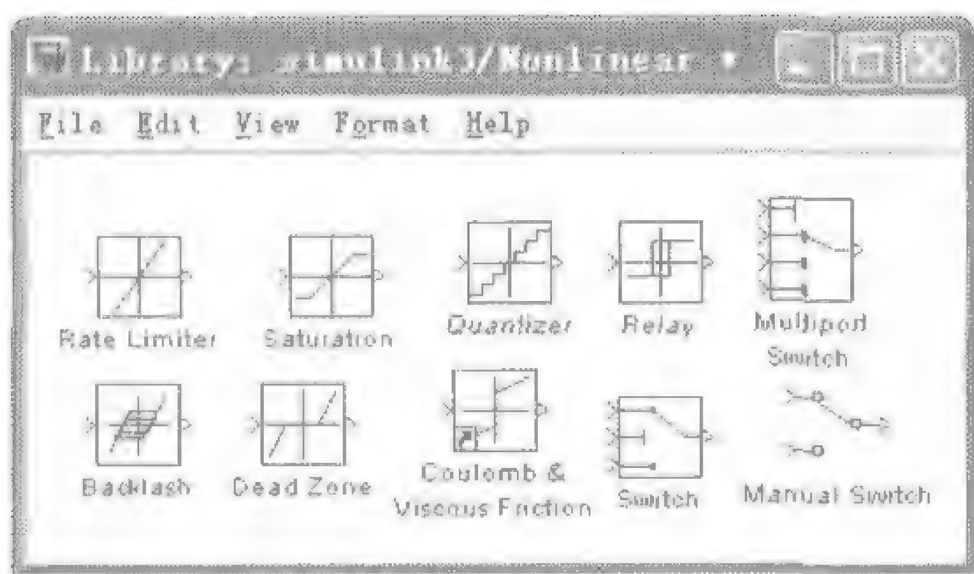


图 8-27 非线性模块库模块

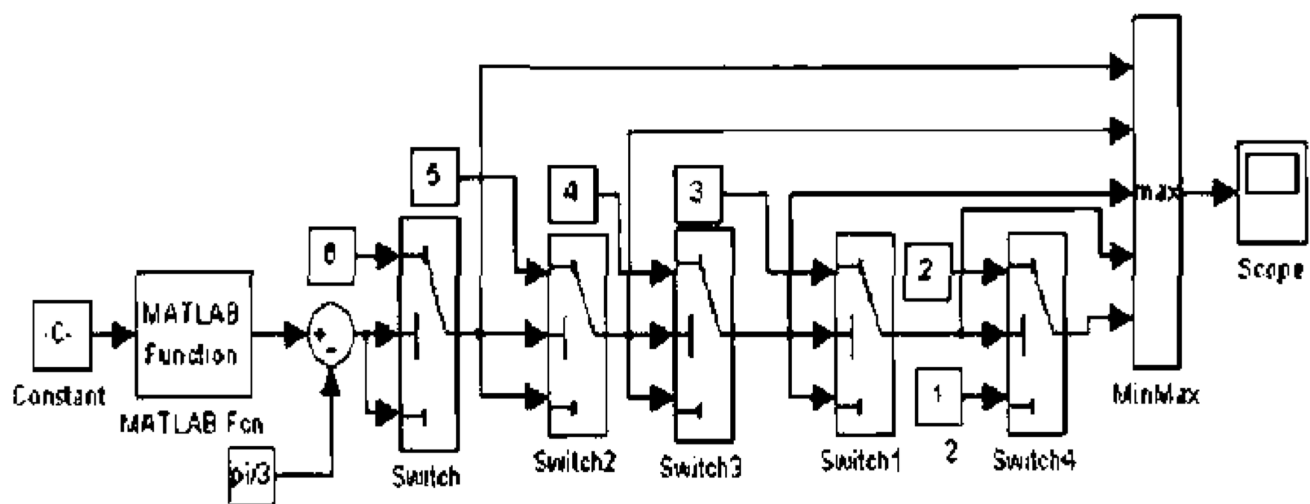


图 8-28 区间选择仿真模型

表 8-4 角度区间查询

角 度	0～π/3	π/3～2π/3	2π/3～π	π～4π/3	4π/3～5π/3	5π/3～2π
区间号	1	2	3	4	5	6

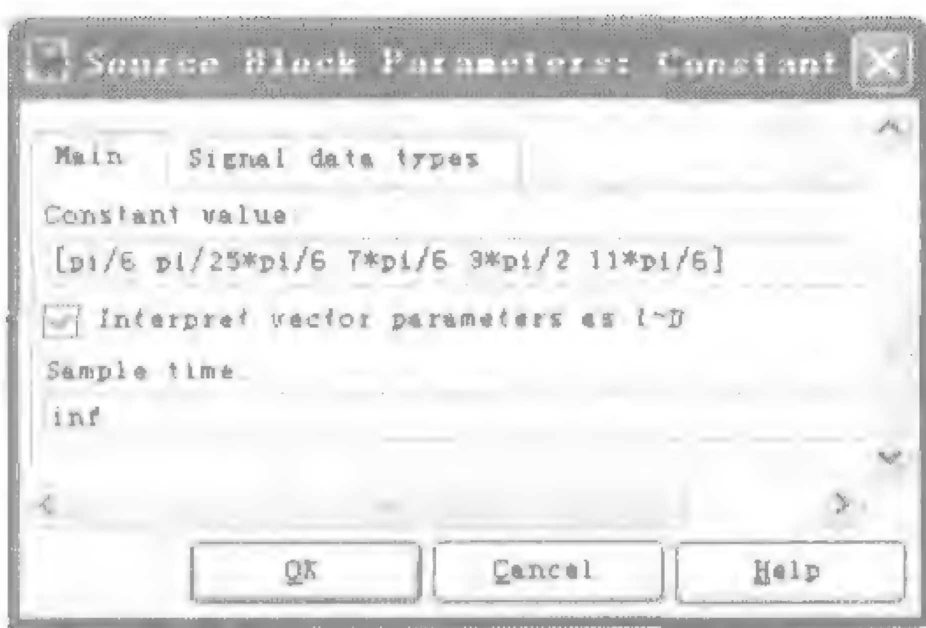


图 8-29 常数信号输入属性页设置



图 8-30 MATLAB Fun 模块属性页设置

区间查表仿真结果如图 8-31 所示。

当然，对于这样一个简单的角度区间查询的功能实现，也可以直接编写 MATLAB 的 M 文件进行仿真，可以不需要搭建如此复杂的仿真模型结构。编写程序代码如下。

```
function num=positionDetection(u)
u=mod(u,2*pi);
if (u>=0) & (u<pi/3)
    num=1;
elseif (u>=pi/3) & (u<2*pi/3)
    num=2;
elseif (u>=2*pi/3) & (u<pi)
    num=3;
elseif (u>=pi) & (u<4*pi/3)
    num=4;
elseif (u>=4*pi/3) & (u<5*pi/3)
    num=5;
else
    num=6;
end
```

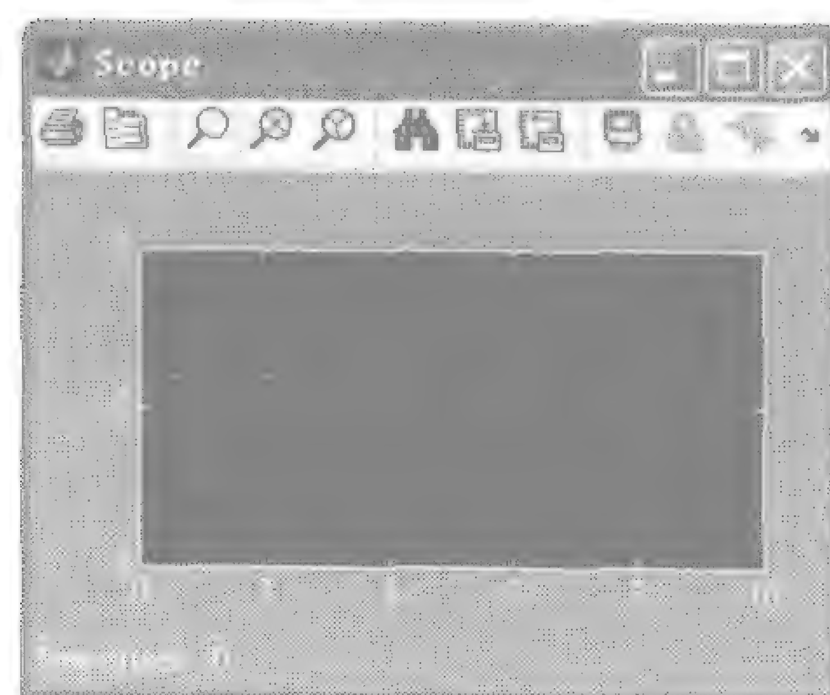


图 8-31　区间查表仿真结果

选择 MATLAB Fcn 模块，在属性页下的 MATLAB function 下输入自定义的 M 文件名 positionDetection，那么同样可以实现如表 8-4 所示的角度区间查询功能。通过两种方法的比较，得知编写 M 文件同样可以实现很多复杂的功能，毕竟用 M 文件编写程序更加灵活。

编写 M 文件的格式：function Output=DefinationFunction (Input)，这里输入 Input 和输出 Output 变量都可以是一维或者是多维的。当输入输出变量为多维情况时，可以用 Signal Routing 下的 Mux 和 Demux 模块进行信号的综合和分解。综合地运用 Simulink 模块库、M 文件编程，可以更加灵活地建立复杂的仿真模型。

8.4.4　信号和系统模块库

信号与系统模块主要实现数据总线，信号综合和分解、信号选择、数据存储、读写等功能。基本组成模块如图 8-32 所示。表 8-5 给出了信号和系统模块库各模块的名称和功能描述。

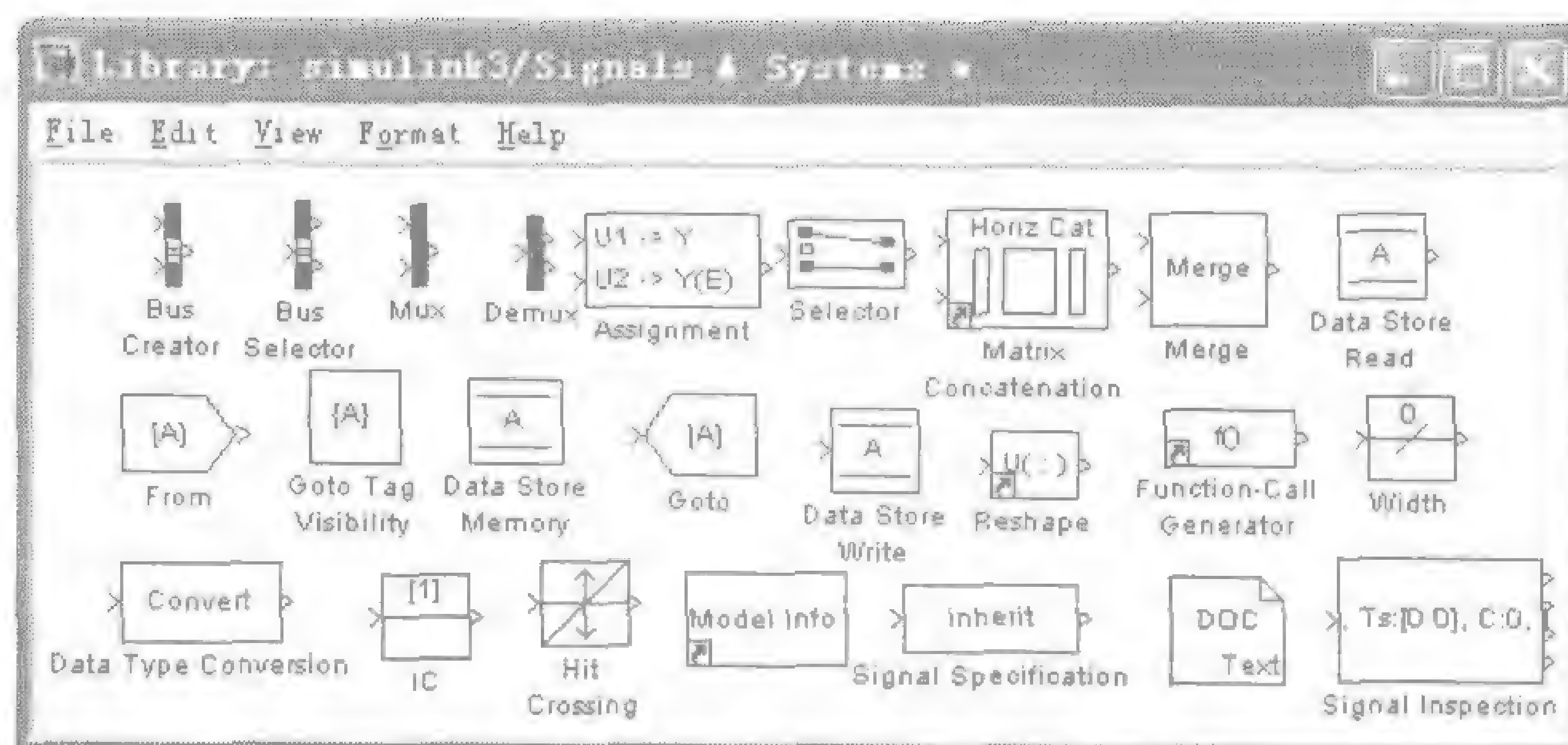


图 8-32　信号与系统模块库

表 8-5 信号与系统模块库模块名称和功能描述

模块名称	功能描述
Bus Creator	创建一个信号总线
Bus Selector	从输入总线中选择输出信号
Mux	将多个输入信号综合成向量输出
Demux	将向量输入信号分解成多个信号输出
From	接受从 Goto 模块的输入
Goto	将模块输出传递到 From 模块
Goto Tag Visibility	定义 Goto 模块标志的实用范围
Selector	选择输入向量信号或者对输入信号重新排序
Assignment	向给定信号(向量或者矩阵)的某元素赋值
Matrix Concatenation	将输入信号连接成列向量、行向量或者矩阵
Merge	将输入信号合并成一个信号输出
Date Store Read	从共享的数据内存区中读出数据
Date Store Memory	定义一个共享的数据内存区
Data Store Write	向共享的数据内存区中写入数据
Reshape	改变输入信号的维度
DataType Conversion	对输入信号数据类型进行转换
Hit Crossing	检测过零点
IC	设置输入信号的初始值
Width	输出输入向量的宽度
Model Info	在模块中显示仿真模型版本信息，包括作者、修改日期、模型描述、模型版本等
Signal Specification	设置信号的属性
DOC Text	在模型中创建 txt 文件

信号与系统模块库主要包括各种信号流向和总线设置的一些模块，以及数据内存 Data Store Memory 相关模块、数据类型转化 Data Type Conversion 模块。

8.4.5 数学模块库

数学模块库是 Simulink 中较为核心的模块库，在实际的仿真模型中，仿真对象总是可以用数学模型来描述，因此涉及各种数学运算。同时数学模块库也是构建自定义模块的基础。

从某种意义上说，利用数学模块库可以建立任何一个可以用数学模型表示的仿真对象。因此数学模块库也是 Simulink 仿真中使用最为频繁的模块库。

如图 8-33 所示为数学模块库中的基本模块。表 8-6 给出了各模块的名称和功能描述。数学模块库几乎包含了常用的基本数学操作，包括加减乘除运算、增益运算、三角函数、取整函数、逻辑比较操作、复数操作等相关函数。对于一个给定数学模型的实际物理系统，都可以通过这些数学运算模块来进行搭建仿真，读者应该要熟练掌握这些模块的使用。

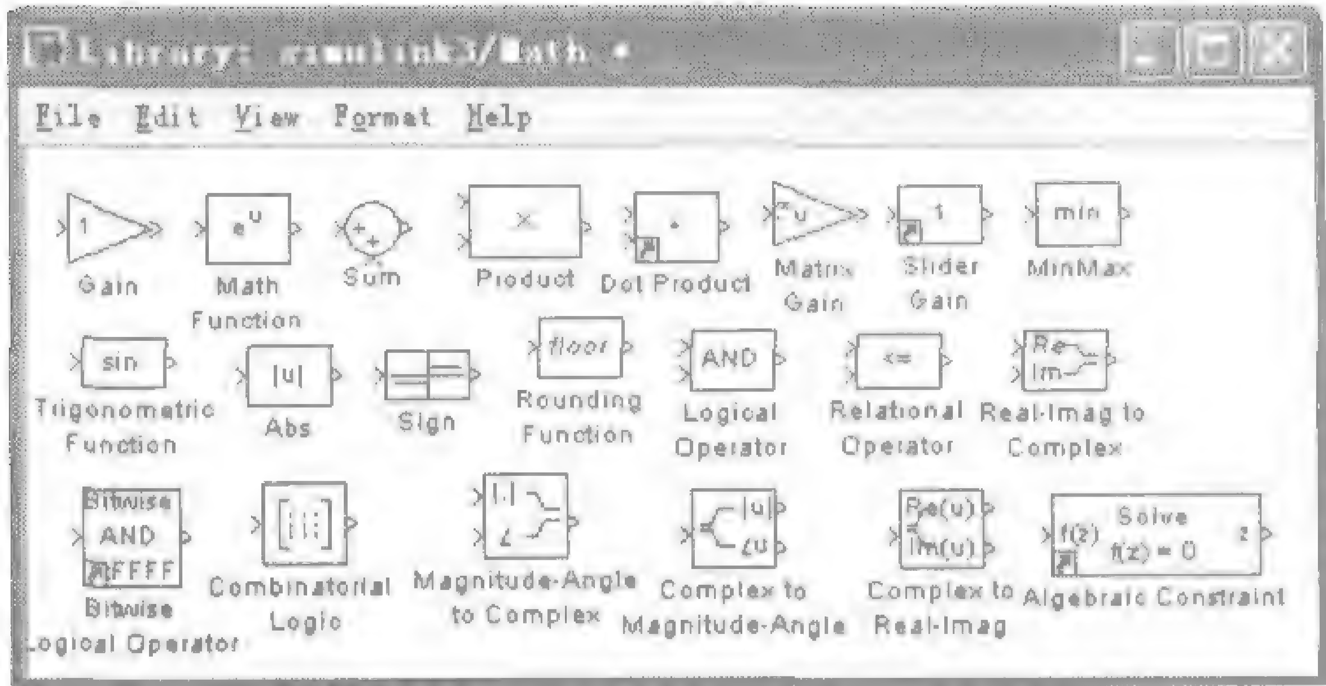


图 8-33 数学模块库

表 8-6　数学模块库模块名称和功能描述

模块名称	功能描述
Sum	对输入信号进行相加减
Product	将输入信号相乘或相除
Dot Product	对输入信号(向量或者矩阵)点乘
Combinatorial Logic	输出输入信号(布尔逻辑)对应行的组合逻辑
Logical Operator	逻辑操作符，包括与、或、非等
Bitwise Logical Operator	位逻辑运算符
Relational Operator	关系运算符
Gain	将输入信号(向量或者单值)乘上一定倍数输出
Slider Gain	滚动条增益
Matrix Gain	将输入信号(矩阵)乘上一定倍数输出
Complex to Magnitude-Angle	将复数转化为幅值和相角形式
Magnitude-Angle to Complex	将幅值和相角形式转化为复数
Complex to Real-Imag	将复数转化为实部和虚部形式
Real-Imag to Complex	将实部和虚部形式转化为复数
Math Function	数学函数
Trigonometrix Function	三角函数
MinMax	输出输入信号的最小或者最大值
Abs	输入信号取绝对值输出
Sign	符号函数
Rounding Function	取整函数
Algebraic Constraint	限制输入信号为 0

8.4.6　输出模块库

输出模块库主要用于数据图像化显示、仿真数据的保存显示、仿真中止等模块。其中示波器模块和输出端口 Out1 模块使用较为频繁。如图 8-34 所示为输出模块库中的基本模块。表 8-7 所示为各模块的名称和功能描述。

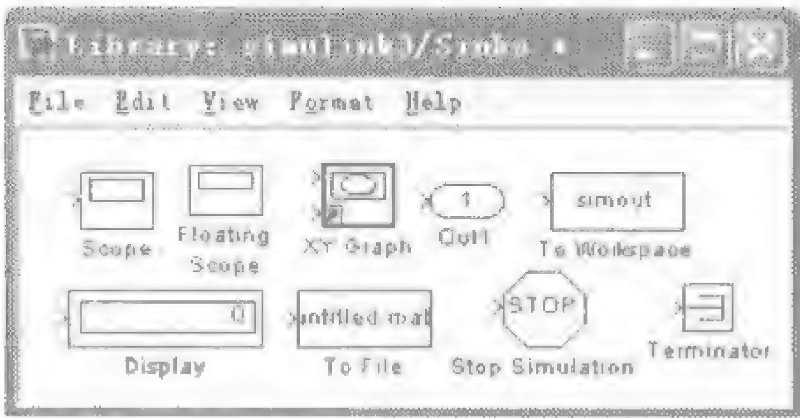


图 8-34　输出模块库

表 8-7　输出模块库模块名称和功能描述

模块名称	功能描述
Scope	示波器，数据图像化显示
Floating Scope	浮点示波器
XY Graph	XY 轴图形显示
Out1	输出端口
Display	数字显示

续表

模块名称	功能描述
To File	输出数据到 mat 文件
To Workspace	输出数据到工作窗口
Terminator	无连接端口的终端
Stop Simulation	当输入非零时停止仿真

8.4.7　输入源模块库

输入源模块库同样是 Simulink 中使用较为频繁的模块库，主要用于产生仿真所需要的输入信号源，包括阶跃信号、斜坡信号、正弦信号、随机数信号、仿真时钟等。如图 8-35 所示为输入源模块库中的基本模块。表 8-8 所示为各模块的名称和功能描述。

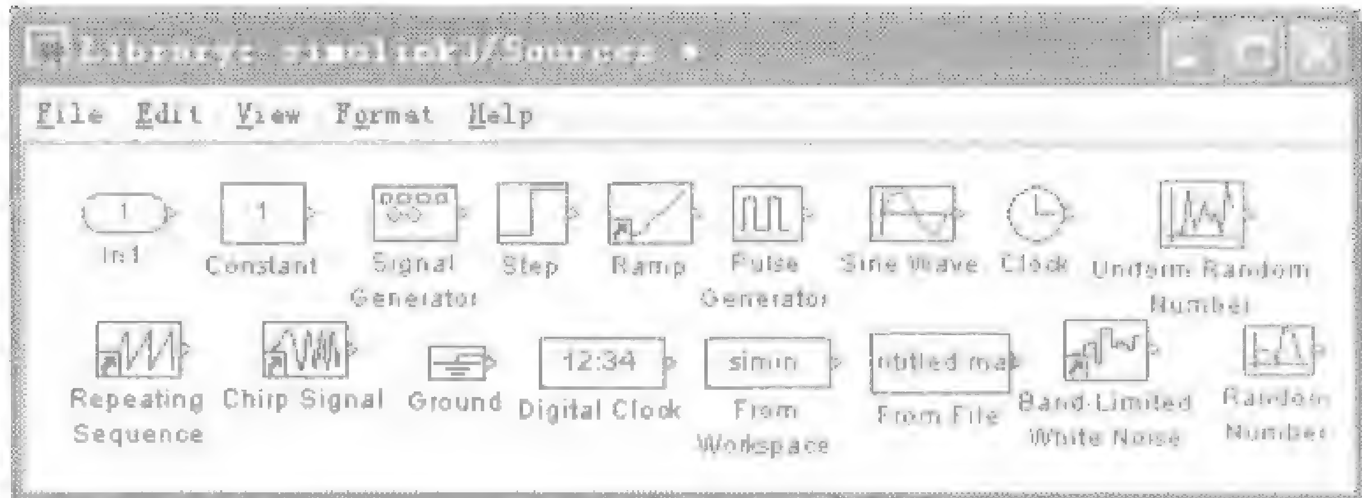


图 8-35　输入源模块库模块

表 8-8　输入源模块库模块名称和功能描述

模块名称	功能描述
In1	输入端口
Constant	产生常数
Signal Generator	信号发生器，可产生正弦波方波、锯齿波和随机数
Ramp	产生斜坡信号
Pulse Generator	脉冲发生器
Step	产生阶跃信号
Repeating Sequence	产生重复序列
Sine Wave	产生正弦波信号
Chirp Signal	产生噪声信号
Ground	无连续信号的接地
Clock, Digital Clock	仿真时钟，数字时钟
From File	读入来自文件的数据
From Workspace	读入来自工作窗口的变量数据
Random Number	产生随机数
Uniform Random Number	产生均匀随机数
Band-Limited White Noise	产生白噪声信号

输入源模块库是实现数据输入、输出、信号的产生以及仿真结果的显示。总结起来，MATLAB 数据的输入输出有以下的几种方法。

1）数据的输入

（1）从工作空间中输入数据：选择 Sources 模块库下的 From Workspace 模块，需要定义从工作空间中输入数据的名称。

（2）从外部输入数据文件：选择 Sources 模块库下的 From File 模块，定义数据文件的名称，注意数据文件格式必须是.mat 文件，如果不是，那么用户可将输入数据文件保存为.txt 文件，

然后加载到工作空间，最后使用 save 命令保存这个工作空间的数据变量。同时.mat 文件中必须包含时间数据，否则仿真出错。

（3）从 From 端口中输入数据：选择信号与系统模块库的 From 端口，选择数据的来源端口名称，即 Goto 模块中端口的定义。

（4）从数据内存中读入数据：选择信号与系统模块库中的 Data Store Read 模块，定义内存数据的名称。

2）数据的输出

（1）将数据输出到工作空间：选择 Sinks 模块库下的 To Workspace 模块，需要定义仿真输出数据的文件名。

（2）将数据输出到数据文件：选择 Sinks 模块库下的 To File 模块，需要定义输出数据文件的名称。

（3）将数据输出到 Goto 端口：选择信号与系统模块库的 Goto 端口。

（4）将数字写入数据内存单元：选择信号与系统模块库中的 Data Store Write 模块，同时必须选择 Data Store Memory 模块以提供数据保存的内存。

上面提供的 4 组数据输入输出方法中，前 3 组方法可以独立使用，但是后一种方法，必须是一一对应，即有相应的 From 端口，就必须存在 Goto 端口与之相对应，否则无法找到数据源。

8.5 Simulink 的命令行仿真技术

在前面所有小节中，系统模型的建立、仿真和分析都是基于 Simulink 可视化图形界面窗口上进行的，由于 Simulink 平台建立了丰富的 Simulink 模块库和其他系统模块库，在 Simulink Library Browser 窗口下简单地拖动模块，进行模块连线、模块属性设置等，就可以进行动态系统的仿真。

这种图形化的操作界面确实给仿真模型的建立分析带来了极大便利，但在一些特殊场合下，还需要进行 Simulink 命令行仿真技术，即使用 MATLAB 命令行对模型进行仿真。比如用户需要对模型中的参数进行动态的改变，或者需要对模型进行多次重复的仿真，并且记录每次仿真的结果，以便后续的分析。如果此时用户每修改一次参数，然后进行仿真，那么势必耗费大量的时间和精力守在电脑前。Simulink 命令行仿真技术为这些特殊要求提供了解决的方案，通过命令行仿真，可以实现动态改变模型参数，进行多次模型的重复仿真和数据分析等。

8.5.1 命令行创建 Simulink 仿真模型

从广义上说，凡是涉及命令行调用实现 Simulink 仿真的 MATLAB 命令，都可以称为是命令行 Simulink 仿真。MATLAB 提供了丰富的命令与 Simulink 工具箱进行接口，方便进行命令行创建和 Simulink 模型仿真。在介绍 Simulink 命令仿真技术之前，阅读下面一段程序

```
%创建模型，并设置参数，进行仿真
new_system('command_sim_demo2')
open_system('command_sim_demo2')
add_block('built-in/step','command_sim_demo2/step')
add_block('simulink/Continuous/Transfer Fcn','command_sim_demo2/2-order system')
add_block('built-in/scope','command_sim_demo2/scope')
add_line('command_sim_demo2','step/1','2-order system/1')
```

```
add_line('command_sim_demo2','2-order system/1','scope/1','autorouting','on')
save_system('command_sim_demo2','command_create_demo2')
%设置模块参数和仿真参数
set_param('command_create_demo2/step','Time','0','Before','0','After','1','SampleTime','0')
set_param('command_create_demo2/2-orde system','Numerator','[100]','Denominator','[1 141.4 100]');
set_param('command_create_demo2/scope','NumInputPorts','1')
set_param('command_create_demo2','solver','ode45','StopTime','10')
%保存后进行仿真
save_system
```

相信读者通过一些函数的命令和程序注释已经明白了这段程序的功能。创建文件名为 command_sim_demo2 的仿真模型，并且将其保存为 command_create_demo2。然后设置模块参数和仿真参数，该仿真模型包含一个阶跃信号源、传输函数模块和示波器模块，阶跃信号模块设置阶跃时间为 0，阶跃值为 1，传递函数模块分子多项式为 100，而分母多项式为[1 141.4 100]，设置示波器的输入端口数目为 1 个，最后设置模型的仿真参数，采用变步长 ode45 算法，仿真结束时间为 10s。经过模块参数和仿真参数的设置后，保存仿真模型，并进行仿真。

如图 8-36 所示为命令行建立的仿真模型，而图 8-37 为仿真结果。从这个实例可以看出，实际上用命令来创建仿真模型是比较烦琐的事情，对于如图 8-36 所示的简单仿真模型，需要编写大量的程序代码，而且需要用户熟悉各个模块所在 Simulink 的位置。

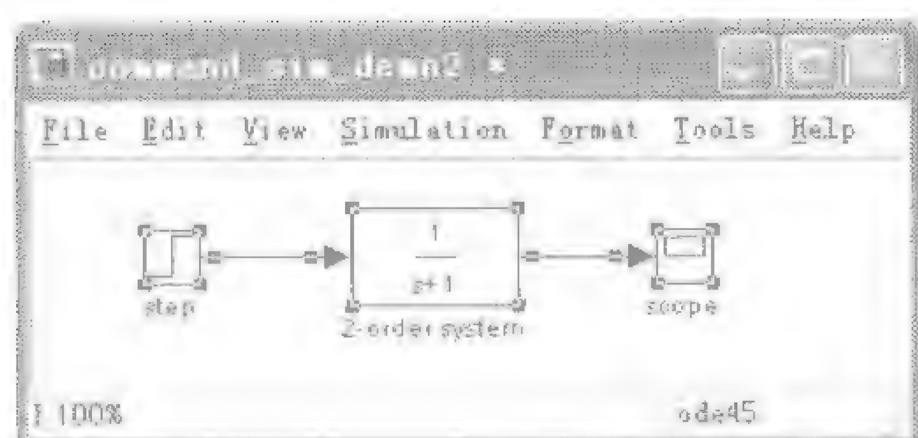


图 8-36　命令行建立仿真模型

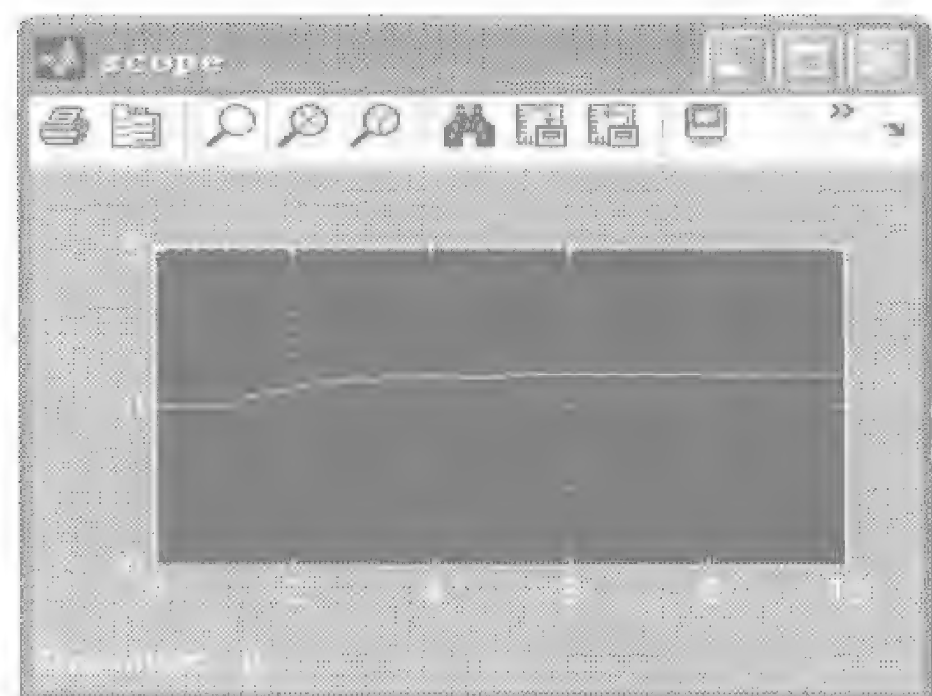


图 8-37　仿真结果

虽然可以使用 Build-in 命令调用 Simulink 任何模块库中的文件，但是用户还必须熟悉各模块的名称，这对于用户而言，要记住每个模块的名称，显然很不现实，而且确实也没有这样的必要性。

通过图形化的 Simulink 仿真平台进行模块的拖放和连线、属性设置是比较方便实用的。因此在本小节中将不会详细地介绍命令行创建模型的指令，在实际应用中，也不会使用这些指令进行仿真模型的创建，上面的程序只是向读者演示一下如何使用命令行创建和仿真模型，以便读者建立命令行仿真技术的基本概念。但是 set_param 和 get_param 两条命令在后续命令行仿真技术介绍中使用比较广泛，因此对于这两条 Simulink 仿真命令做详细的介绍。

1）set_param 命令

set_param 命令用来设置仿真模型的参数或者仿真模型中具体模块的参数。具体的调用格式如下。

```
%设置仿真模型或者模块的参数
set_param('obj','parameter1',value1,'parameter2',value2,…)
%将仿真模型参数设置为默认值
set_param(0,'modelparm1',value1,'modelparm2',value2,…)
```

第一条指令允许用户设置仿真模型的属性，包括仿真算法设置等，同时也允许用户设置具体仿真模块的参数。通过使用 get_param 命令来获取仿真模块属性名称和仿真算法属性名称。

第二条指令允许用户将仿真模型参数设置为默认值，即改变 MATLAB 默认的 Simulink 仿真参数，在新建仿真模型后，那么模型的默认参数值就是用户自己设定的 Simulink 仿真参数。为了获取仿真参数，可以使用 simset 命令得到仿真模型的仿真结构参数，包括属性名称和允许的属性值。

```
%设置传递函数模块，分子多项式为[100]，分母多项式为[1 141.4 100]
set_param('command_create_demo2/2-order system','Numerator','[100]','Denominator','[1141.4 100]');
%设置仿真模型的仿真参数
set_param('command_create_demo2','solver','ode45','StopTime','10')
%设置仿真模型默认参数：采用 ode23 算法，初始步长为 le-3
set_param(0,'solver','ode23','InitialStep','le-3')
%设置仿真模型的位置
set_param(' command_create_demo2/step','Position',[50 100 110 120])
%设置仿真模块的回调函数
set_param(' command_create_demo2/step','OpenFcn','open_fcn','CloseFcn','close_fcn')
```

注意

模块或仿真模型参数属性值通常是字符串形式，从上面的实例中可以看到 Position 属性的属性值是数组，另一个非字符串属性值的属性 UserData 同样是用户自定义数据类型。

2）get_param 命令

get_param 命令可以获取指定模块的属性值、当前仿真模型的默认参数以及仿真模块属性的结构体，即模块的属性名称。具体的调用格式如下。

```
%获取仿真模型参数和特定仿真模块参数属性值
get_param('obj','parameter')
%获取多个模块参数属性值，模块路径以{}元胞形式定义
get_param({object},'parameter')
%返回句柄 handle 对象的参数属性值
get_param(handle,'parameter')
%获取当前仿真模型或者模块的默认属性值
get_param(0,'parameter')
get_param('obj','ObjectParameter')   %获取对象参数的结构体
get_param('obj','DialogParameter')   %获取仿真模型的参数名称结构体
```

以下通过具体的命令形式和注释演示这些命令的调用形式。

```
>> get_param('command_create_demo2','Solver')
ans=
ode45
>> get_param('command_create_demo2/step','SampleTime')
%获取 step 模块和 scope 模块的对话框属性名称结构体
>> param=get_param({'command_create_demo2/step','command_create_demo2/scope'},
'DialogParameters')
param =
```

```
    [1×1 struct]
    [1×1 struct]
%可以分别使用 param{1}和 param{2}查看 step 模块和 scope 模块的对话框属性名称
%查看仿真模型中所包含的模块类型
>> blocks=find_system(gcs,'Type','block')
>> listblks=get_param(blocks,'BlokType')
listblks =
    'TransferFcn'
    'Scope'
    'Step'
%获取 step 模块对话框参数
>> get_param('command_create_demo2/step','DialogParameters')
ans =
        Time: [1×1 struct]
        Before: [1×1 struct]
        After: [1×1 struct]
        SampleTime: [1×1 struct]
        VectorParamslD: [1×1 struct]
         zeroCross: [1×1 struct]
```

8.5.2　用 Simulink 命令行进行仿真

前一小节简单介绍了使用命令行创建 Simulink 仿真模型的过程，可以看出对于一个非常简单的 Simulink 仿真模型，如果模块选择、模块间的连线、模块参数设置和仿真模型参数设置等一系列的操作完全使用命令行来完成时，编写相应的程序代码将非常烦琐，因此不建议使用命令行来创建 Simulink 仿真模型，而直接使用 Simulink 仿真平台的图形可视化界面，在 Simulink Library Browser 窗口中进行模块的拖放、连线、属性设置、仿真参数设置等，这样的过程是比较简单实用的。

Simulink 命令行仿真的优势在于能够重复地进行仿真模型的仿真，动态地改变仿真模型和模块的参数，记录多次仿真模型的结果，并进行数据分析。本小节将主要介绍一些非常实用的 MATLAB 命令行模型动态仿真技术。

在介绍相关的命令行仿真技术前，我们通过一个实例向读者演示一些仿真技巧，仿真模型采用如图 8-36 所示的仿真模型。仿真模型中 Step 模块和 State-Space 模块的参数设置如图 8-38 和图 8-39 所示，由于采用命令行进行仿真，相关的参数以变量形式表示，可以动态地改变仿真过程的模块参数。仿真过程需要对每次仿真结果保存，那么需要设置示波器的属性如图 3-40 所示，保存变量名为 Sim_out，数据格式为 Structure with time，这样就可以记录保存每次仿真过程的仿真时间和系统输出。



图 8-38　Step 模块的参数设置

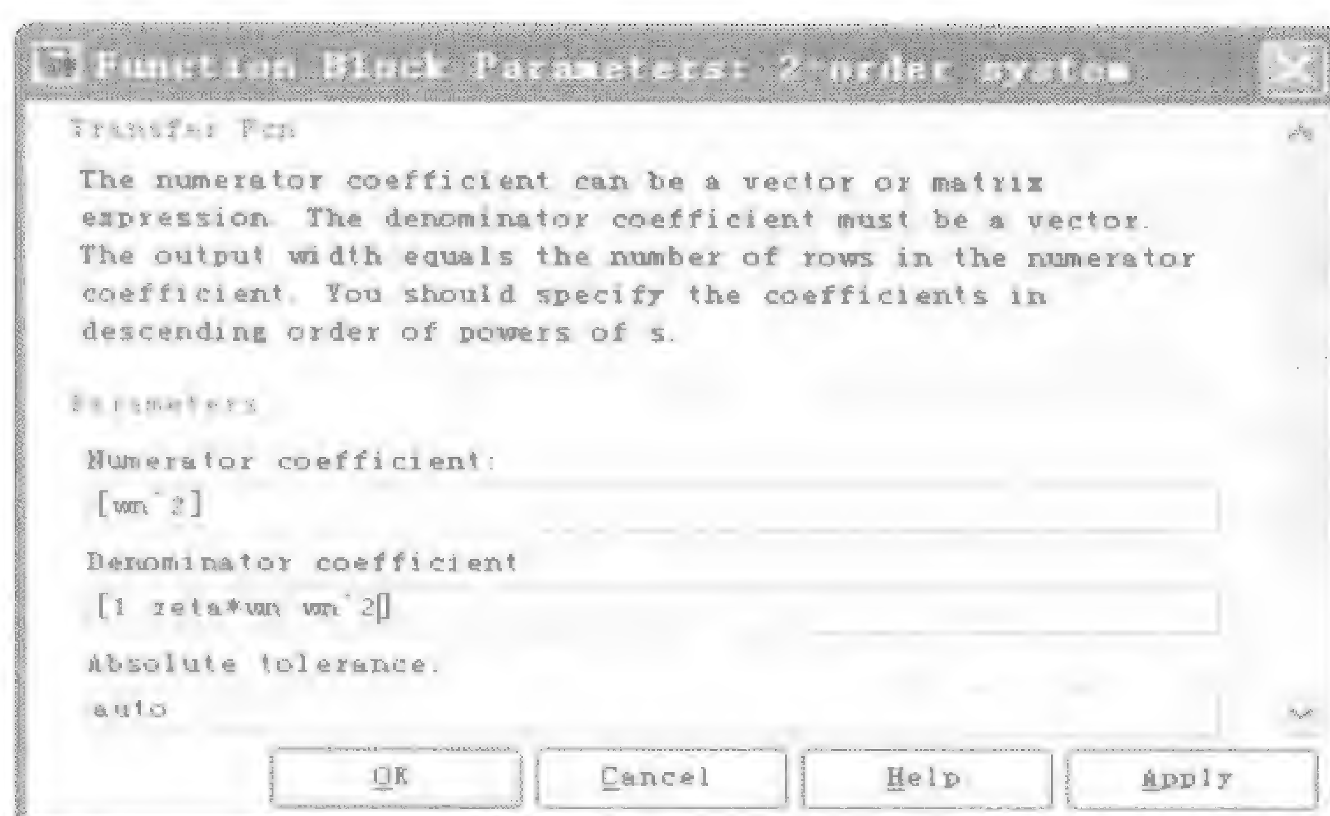


图 8-39 State-Space 模块的参数设置

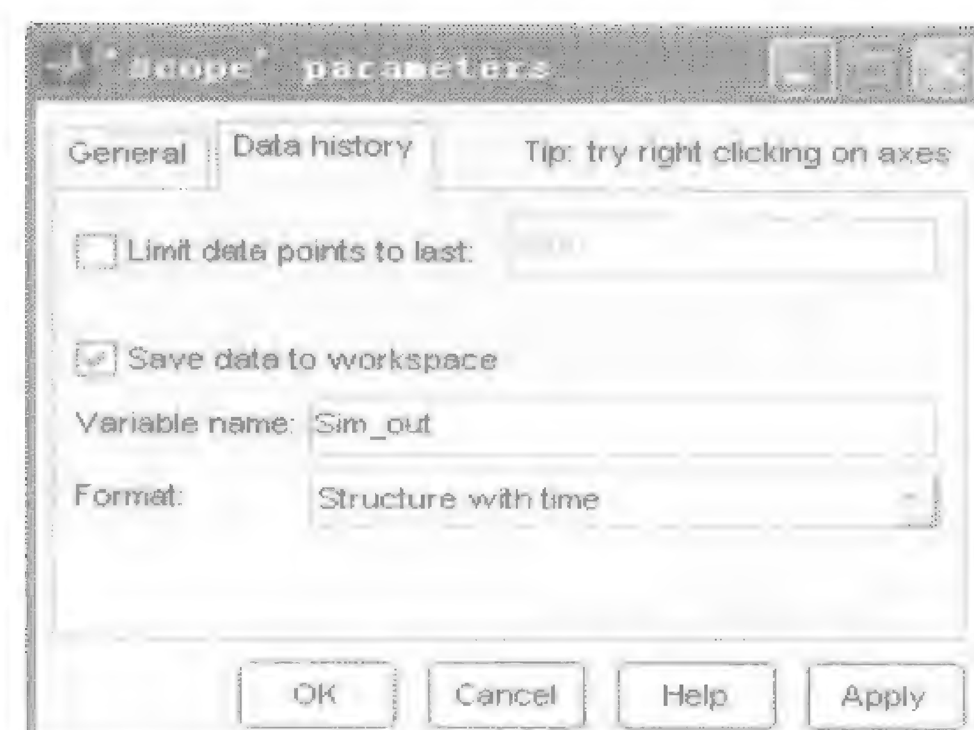


图 8-40 示波器模块属性设置

建立好如图 8-36 所示的仿真模型后，需要使用命令行仿真技术实现在 zeta=0.707、wn=[4,6,8,10]时对应的系统输出，并将它们同时画在一个图上，比较系统输出的差异。

```
Tstart=0;
final_value=1;
zeta=0.707;
wn_ref=[4 6 8 10];
%设置仿真模型的算法和终止时间
set_param('command_create_demo2','solver','ode45','StopTime','4');
% set_param('command_create_demo2/step','Time','0','After','1');
for i=1:length(wn_ref)
    wn=wn_ref(i);
    sim('command_create_demo2');
    time{1,i}=Sim_out.time; %保存仿真时间
    out_value{1,i}=Sim_out.signals.values; %保存输出结果
end
%结果对比显示
colorStyle={'r-','k-','b-','m-'};
for i=1:length(time)
    plot(time{1,i},out_value{1,i},colorStyle{i});
    hold on;
end
legend('wn=4rad/s','wn=6rad/s','wn=8rad/s','wn=10rad/s');
```

仿真结果如图 8-41 所示。在 Simulink 仿真中，仿真模型可以直接从 MATLAB 的 Workspace 中调入仿真参数进行仿真。从这个实例中，读者可以体会到：当仿真过程模型参数动态变化时，利用命令仿真技术可以非常简单、快捷地实现模型参数动态变化、仿真结果的对比分析等。以这样一个实例的引入，接下来向读者详细介绍一些实用的 Simulink 命令仿真技术。

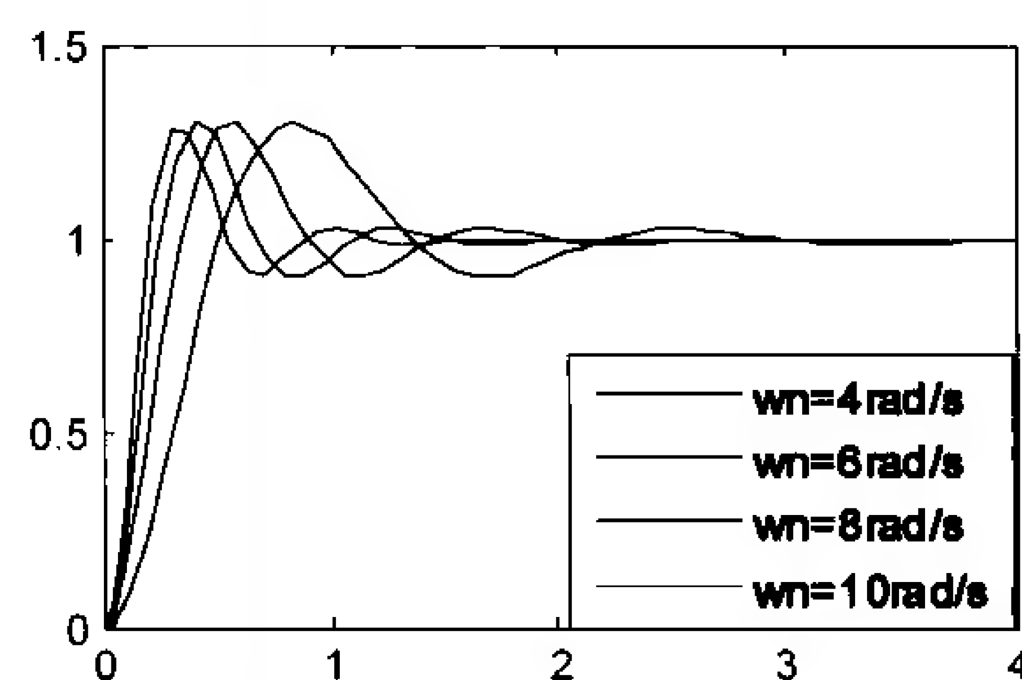


图 8-41 命令行仿真结果

在使用命令行进行动态系统仿真中，使用较为频繁的几条命令包括 sim、simset、simget 和 simplot。下面分别介绍这几条命令的含义和使用方法。

1）sim 命令

sim 命令可以对指定的系统模型按照给定的仿真参数和模型参数进行动态仿真。命令的调用格式如下。

```
[t, x, y]=sim(model, timespan, options, ut);
[t, x, y1, y2, ..., yn]=sim(model, timespan, options, ut);
```

函数输入输出参数说明：以上两条命令是 sim 指令完整的调用模式，在通常情况下，除了 model 参数外，其他参数都可以是空集，使用系统模型的默认属性设置，包括系统配置参数和模型参数。Model 参数为指定的仿真模型名称。

timespan 为仿真时间设置选项，可以有以下几种仿真时间设置方法。

（1）timespan=tFinal，设置终止时间 tFinal，起始时间默认为 0。

（2）timespan=[tStart tFinal]，设置仿真模型的起始时间 tStart 和终止时间 tFinal。

（3）timespan=[tStart OutputTimes tFinal]，设置仿真模型的起始时间 tStart 和终止时间 tFinal，同时设置仿真过程中输出的时间向量 OutputTimes。

前两种仿真模型的时间设置输出的时间向量由仿真算法和仿真模型来确定，而第三种时间设置规定了仿真模型在哪些点上进行仿真输出。

options 选项包括了除仿真时间外的所有模型参数的设置，包括仿真算法选择、步长设置、相对误差和绝对误差设置、输入输出数据名称和格式设置等选项，由 simset 进行设置。

ut 为外部变量的输入，可以是多个外部变量，格式是 $n\times2$ 矩阵，第一列对应输入变量的时间，第二列对应输入变量的值。

输出变量包括系统仿真时间向量 t、系统仿真状态变量矩阵 x 和系统仿真的输出矩阵 y。

```
%采种所有默认仿真参数和时间仿真模型
>> [t, x, y]=sim ('command_create_demo2')
%设置仿真终止时间为 4s，仿真输出时间向量由默认求解算法决定
>> [t, x, y]=sim ('command_create_demo2',4)
%设置仿真起始时间 0s，终止时间 4s，时间间隔为 0.01s
>> [t, x, y]=sim (' command_create_demo2','[0:0.01:4]')
%设置仿真参数最大步长 le-3，数据保存名称 out_value，数据保存格式 struct
>> options=simget(' command_create_demo2')
>> simset (options, 'MaxStep','le-3','SaveFormat','Structure','OutputVariables','ty')
>> [t, x, y]=sim (' command_create_demo2',[],options)
```

2）simget 命令

同 set_param 命令和 get_param 命令一样，simget 命令和 simset 命令分别表示获取模型除仿真时间外的所有其他参数和设置模型的仿真参数。simget 命令的调用格式为

```
struct=simget (model)
value=simget (model, property)
value=simget (optionStructure, property)
```

第一条指令获取仿真模型的仿真参数结构体，不包含仿真时间参数；第二条指令获取仿真模型指定属性的属性值；第三条指令获取仿真模型结构选项中指定属性的属性值。仿真模型的仿真参数结构体可通过下面的指令获取。

```
options=simget('command_create_demo2')
options =
                        AbsTol: 'auto'
                         Debug: 'off'
                    Decimation: 1
                  DstWorkspace: 'current'
```

```
base、parent
                FinalStateName: ''
                     FixedStep: 'auto'
                  InitialState: []
                   InitialStep: 'auto'
                      MaxOrder: 5
                    SaveFormat: 'Array'
StructureWithTime
                 MaxDataPoints: 1000
                       MaxStep: 'auto'
                       MinStep: 'auto'
                  OutputPoints: 'all'
               OutputVariables: 'ty'
                        Refine: 1
                        RelTol: 1.0000e-003
                        Solver: 'ode45'
                  SrcWorkspace: 'base'
                         Trace: ''
                     ZeroCross: 'on'
                 SignalLogging: 'on'
             SignalLoggingName: 'logsout'
            ExtrapolationOrder: 4
        NumberNewtonIterations: 1
```

3）simset 命令

在进行 Simulink 命令行仿真时，根据用户的不同需求，设置相关的参数进行系统仿真。simset 命令常用的调用格式如下。

```
options=simset (property, value, ...);
options=simset (old_opstruct, property, value, ...);
options=simset (old_opstruct, new_opstruct);
simset
```

从以上 4 条指令可以看出，除了第 4 条指令不返回任何值外，其他 3 条指令返回的均是仿真模型参数属性的结构体。

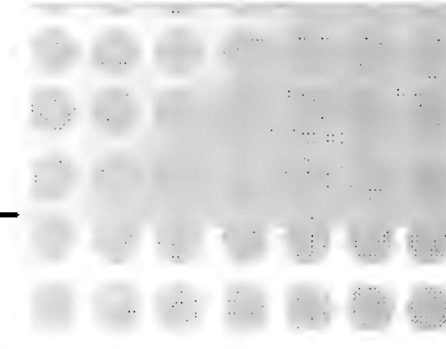
```
%获取当前模型的仿真参数结构体
>> options=simset ('command_create_demo2')
>> solver_method=simget (options, 'solver')
%设置当前仿真参数算法为 ode23，输出变量 out_value 的数据格式为结构体
>> new_options=   simset (options, 'solver', 'ode23', 'SaveFormat', 'Structure', 'OutputVariables', 'ty')
>> [t, x, y]=sim ('command_create_demo2', [], new_options)
```

4）simplot 命令

simplot 命令可以将仿真模型输出结果以示波器的形式绘制出来。其调用格式如下。

```
simplot (data);
simplot (time, data);
```

其中 data 来自仿真模型的输出端口，数据格式可以为 Array、Struct 或者 Struct with time 形式。time 同样可以是 Array 或者 Struct 形式。采用如图 8-36 所示的仿真模型。如图 8-42 所示为



仿真模型命令行仿真结果。

```
%设置仿真模型的算法和终止时间
>> set_param('comand_create_demo2','Solver','ode45','StopTime',4)
%设置阶跃模型参数
>> set_param('comand_create_demo2/step','Time','0','After','1');
>> assignin('base','wn',10);
>> assignin('base','zeta',10);
>> sim('command_create_demo2',[0:1e-5:4]);
>> simplot(tout,yout)
```

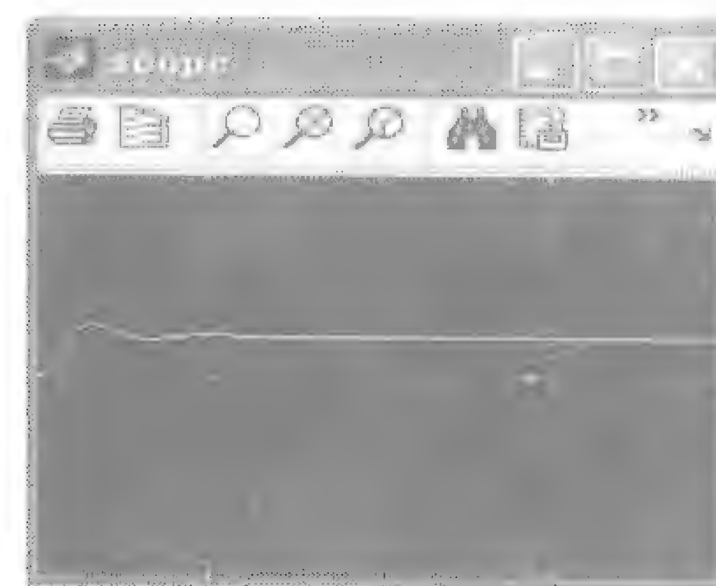


图 8-42 命令行仿真结果

8.5.3 命令行仿真实例

通过前面两节对命令行仿真技术的详细介绍，我们对命令进行仿真技术已有所了解。在这里，将如图 8-36 所示的仿真模型中的输入信号修改为输入端口 In1 模块，以便演示在不同输入情况下系统模型的仿真结果，仿真模型如图 8-43 所示。利用这样一个简单模型，将向读者演示简单模型命令行仿真、不同仿真时间设置、不同模型参数的仿真，以及不同输入信号下模型的仿真。

1．阶跃输入情况下命令行仿真

当输入 ut 为外部输入阶跃信号时，系统仿真模型的命令行仿真结果如图 8-44 所示。

```
assignin('base','wn',10);
assignin('base','zeta',0.7);
options=simget('command_create_demo2');
new_options=simset(options,'Solver','ode45','SaveFormat','Structure','OutputVariables','ty');
t=0:0.001:4;
u=1*ones(1,length(t));
ut=[t',u'];
[tout,x,yout]=sim('command_create_demo2',[],new_options,ut);
```

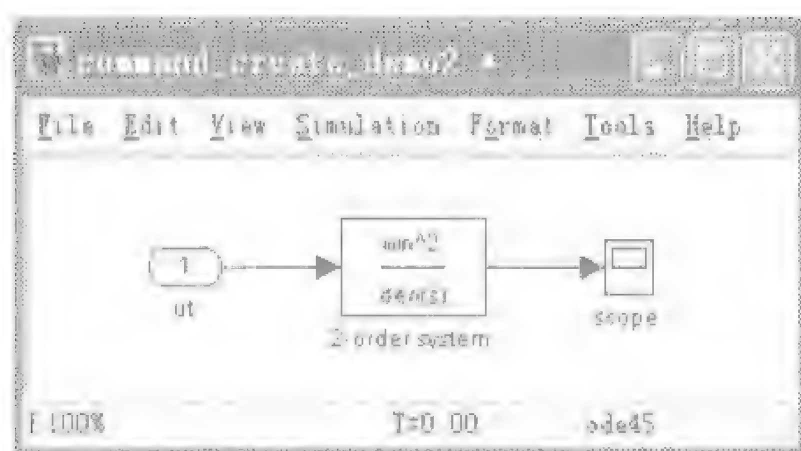


图 8-43 仿真模型

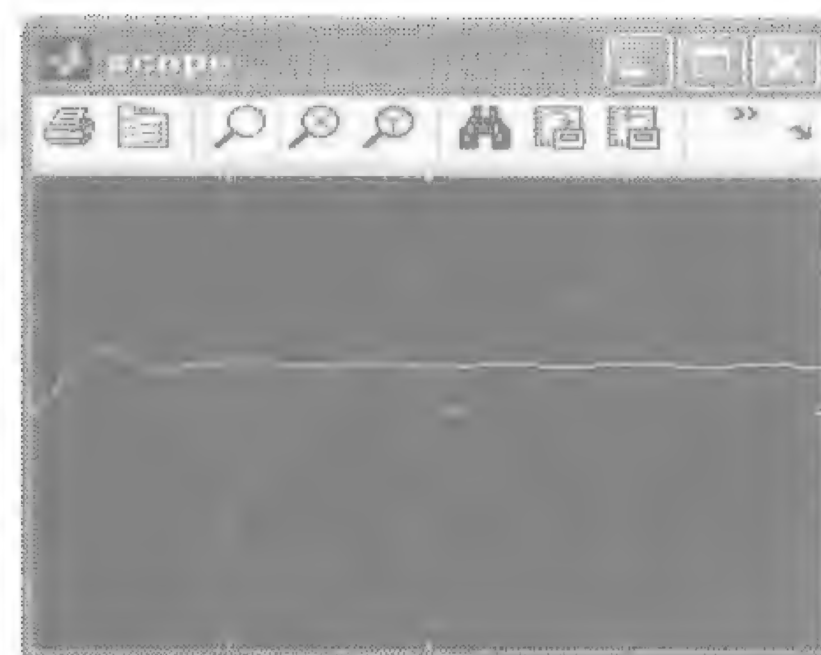


图 8-44 阶跃输入下二阶系统命令行仿真结果

2．Import 端口输入命令行仿真

从如图 8-43 所示的仿真模型中可以看出，输入采用 Import 模块，那么输入信号可以直接从 Workspace 中调入，如图 8-45 所示。选中“Load from workspace”区域下的“Input”选项，可以直接使用[t, u]矩阵或者重新定义新的输入变量名称，但是该变量的值必须也是[t, u]形式的 $n\times 2$ 的矩阵。

```
assignin('base','wn',10);
assignin('base','zeta',0.7);
t=0:0.0001:4;
u=1*ones(1,length(t));
sim_in=[t',u'];
sim('command_create_demo2')
```

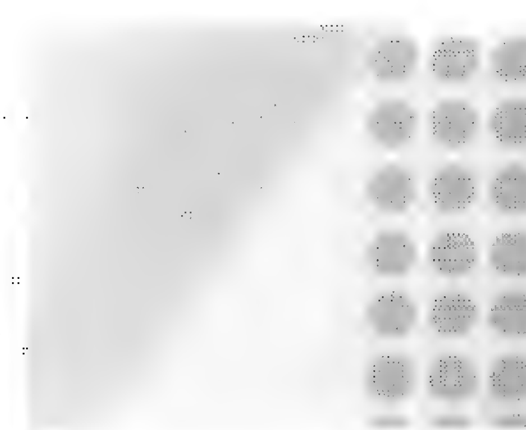




图 8-45　Import 仿真数据输入

3．不同仿真时间的命令行仿真

前面在介绍 sim 命令时，输入参数中 timespan 为仿真时间设置选项，可以有以下几种仿真时间设置方法。

（1）timespan=tFinal，设置终止时间 tFinal，起始时间默认为 0。

（2）timespan=[tStart tFinal]，设置仿真模型的起始时间 tStart 和终止时间 tFinal。

（3）timespan=[tStart OutputTimes tFinal]，设置仿真模型的起始时间 tStart 和终止时间 tFinal，同时设置仿真过程中输出的时间向量 OutputTimes。在这里将分别利用 3 种不同时间对如图 8-43 所示的仿真模型进行命令行仿真。其中示波器数据以 sim_out 变量保存为 Struct with time 形式。其 MATLAB 代码如下。

```
%模块参数设置和模型输入端口设置
assignin('base','wn',10);
assignin('base','zeta',0.7);
t=0:0.0001:10;
u=1*ones(1,length(t));
sim_in=[t',u'];
%仿真时间结构元胞数组
timeSetting={5,[1 5],[0:0.5:4],[0:0.1:4]}
for i=1:length(timeSetting)
    sim('command_create_demo2',timeSetting{i});
    %示波器的结果以变量名 Result 保存在工作窗口下，i=1、2、3、4
    assignin('base',strcat('Result',num2str(i)),sim_out);
end
```

4．不同输入信号下命令行仿真

在很多情况下，用户需要对不同的输入信号来测试系统响应性能情况，那么使用命令行仿真技术，可以不需要重复修改输入信号类型，而直接采用 Import 端口，从 Workspace 中调用输入信号，从而对系统进行仿真，还是使用如图 8-43 所示的仿真模型在输入信号分别是阶跃信号、正弦信号、方波信号和脉冲信号的情况下进行仿真模型的响应情况。其 MATLAB 代码如下。

```
clear
%模块参数设置和模型输入端口设置
assignin('base','wn',10);
assignin('base','zeta',0.7);
```

```
t=0:0.0001:4;
u1=1*ones(1,length(t));
u2=sin(t);
[u3,t3]=gensig('square',1,4,0.001);
[u4,t4]=gensig('pulse',1,4,0.001);
%构建输入信号元胞结构体
ut={[t',u1'],[t',u2'],[t3,u3],[t4,u4]};
for i=1:length(ut)
    %仿真时间和仿真模型属性使用默认设置值
    sim('command_create_demo2',[],[],ut{i});
    assignin('base',strcat('Result',num2str(i)),Sim_out);
end
```

5. 仿真模型模块参数动态变化命令行仿真

在实际高级仿真技术中，仿真模型中一些模块参数动态变化，要求用户得到仿真模型结果随模块参数变化的一些规律。如果用户每修改一次模块参数就仿真一次，并且保存每次的仿真结果，最后综合所有仿真结果进行数据分析，在这种情况下，使用命令行仿真参数可以非常方便地解决模块参数动态变化时模型的仿真问题。其 MATLAB 代码如下。

```
t=0:0.0001:4;
u=1*ones(1,length(t));
wn=10;
zeta_ref=[0.3 0.5 0.707 0.9];
%设置仿真模型的算法和终止时间
set_param('command_create_demo2','Solver','ode45','StopTime','4');
for i=1:length(zeta_ref)
    zeta=zeta_ref(i);
     sim('command_create_demo2',[],[],[t',u']);
     %保存仿真结果和时间
    assignin('base',strcat('Result',num2str(i)),Sim_out);
end
```

8.6 Simulink 应用实例

虽然 Simulink 提供了网络输入函数、传输函数、权值函数等神经网络的基本组件，但并不需要用户在 Simulink 中以这些组件来构造神经网络模型（当然如果不嫌麻烦也可以自己试着建一个），而往往通过 MATLAB 命令窗口或编制程序首先完成网络的设计，然后通过 gensim 函数生成神经网络的仿真模块，并进入 Simulink 系统进行仿真。

仍以第 6 章的例 6-2 中 Elman 神经网络用于峰值检波为例，通过例 6-2，我们已经完成了该问题的网络设计，把该文件另存为 xiu 文件。把存于当前目录中，然后在命令窗口中加载如下文件。

```
>> xiu
>> who
Your variables are:
A    Pq    T    T1    T2    Time    Time1    Time2    Tq    Y    net    p
```

```
>> net
net =
    Neural Network object:
    architecture:
             numInputs: 1
             numLayers: 2
          biasConnect: [1; 1]
         inputConnect: [1; 0]
         layerConnect: [1 0; 1 0]
        outputConnect: [0 1]
        targetConnect: [0 1]
           numOutputs: 1   (read-only)
           numTargets: 1   (read-only)
    numInputDelays: 0   (read-only)
    numLayerDelays: 1   (read-only)

    subobject structures:
               inputs: {1×1 cell} of inputs
               layers: {2×1 cell} of layers
              outputs: {1×2 cell} containing 1 output
              targets: {1×2 cell} containing 1 target
               biases: {2×1 cell} containing 2 biases
         inputWeights: {2×1 cell} containing 1 input weight
         layerWeights: {2×2 cell} containing 2 layer weights

    functions:
             adaptFcn: 'trains'
              initFcn: 'initlay'
           performFcn: 'mse'
             trainFcn: 'traingdx'

    parameters:
           adaptParam: .passes
            initParam: (none)
         performParam: (none)
           trainParam: .epochs, .goal, .lr, .lr_dec,
                        .lr_inc, .max_fail, .max_perf_inc, .mc,
                        .min_grad, .show, .time

    weight and bias values:
                   IW: {2×1 cell} containing 1 input weight matrix
                   LW: {2×2 cell} containing 2 layer weight matrices
                    b: {2×1 cell} containing 2 bias vectors

    other:
             userdata: (user stuff)
```

该网络已经经过了训练和仿真，仿真结果通过例 6-2 呈现出来。从仿真结果上看，输出信

号和调制信号基本吻合，基本完成了调幅信号的峰值检波。

现在的问题是，在 Simulink 环境下如何完成该网络的动态仿真。基本的方法是，在命令窗口加载设计好的神经网络后，以 gensim 生成神经网络仿真模块，由于 Elman 神经网络内部具有延迟单元，所以只能采用离散采样。设离散采样时间为 0.05s，在命令窗口输入下列命令

```
>> gensim(net,0.05)
```

弹出两个窗口，一个窗口是神经网络仿真模型库（Library：neural）窗口，如图 8-46 所示；另一个窗口为 Simulink 系统模型创建窗口，如图 8-47 所示。

可以看出，在该窗口中，已经建立了神经网络模块（Neural Network），另外，还有一个采样输入(p{1})和一个示波器输出(y{1})，将其存为名为 demo.mdl 的仿真文件。

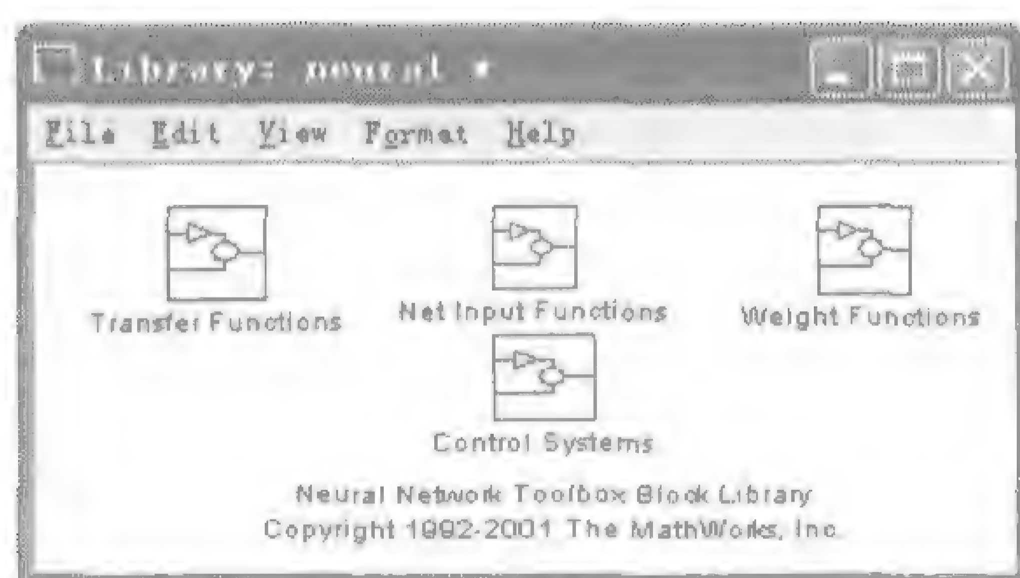


图 8-46 神经网络仿真模型库

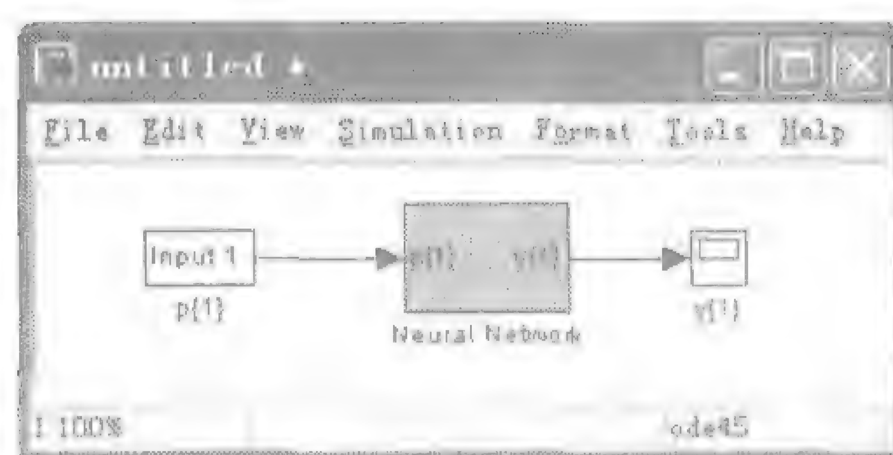


图 8-47 系统模型创建窗口

双击 Neural Network 模块，弹出“untitled/Neural Network”窗口，如图 8-48 所示，显示出网络的详细结构。如果用户还想了解更详细的网络结构，可以在弹出的窗口中双击需要了解的模块，如图 8-49 所示第 1 网络层(Layer1)的结构。这样一直下去，直到出现的窗口为属性设置窗口为止。

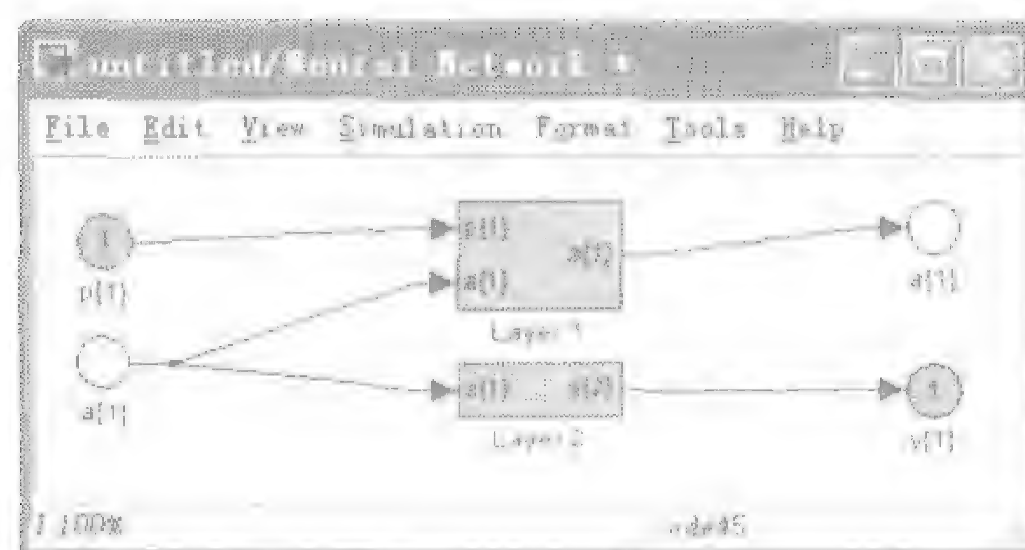


图 8-48 “untitled/Neural Network”窗口

虽然在每个弹出的窗口用户都可以修改和编辑网络结构及其属性，但建议读者最好不要这样做，因为我们建立的神经网络模块是基于命令窗口或程序已经设计好的网络，若在这里进行修改，可能导致网络不能支持其运行，或达不到仿真的预期效果。

如果不做任何修改，对其进行仿真，则输出波形如图 8-50 所示。

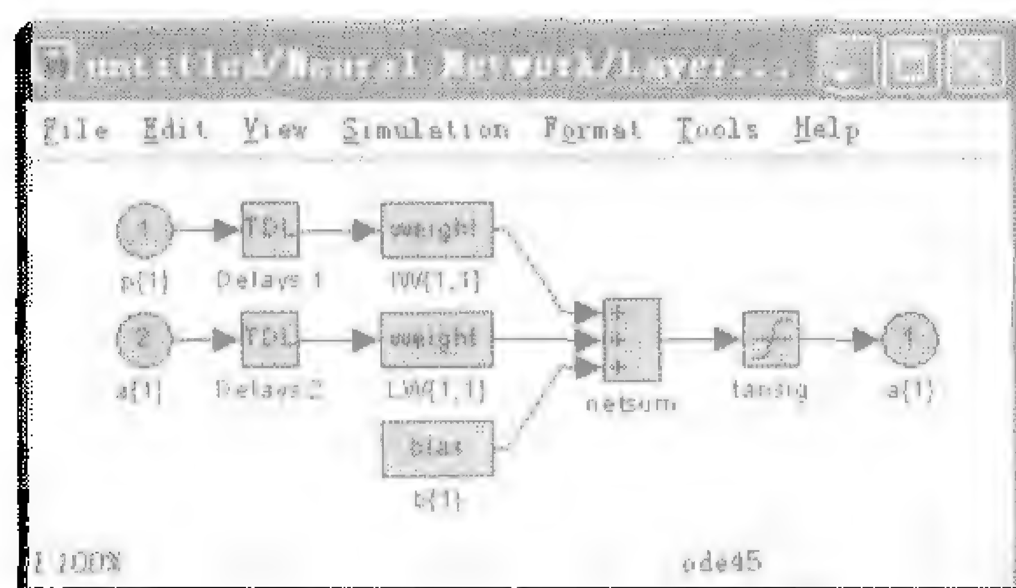


图 8-49 “untitled/Neural Network/Layer1”窗口

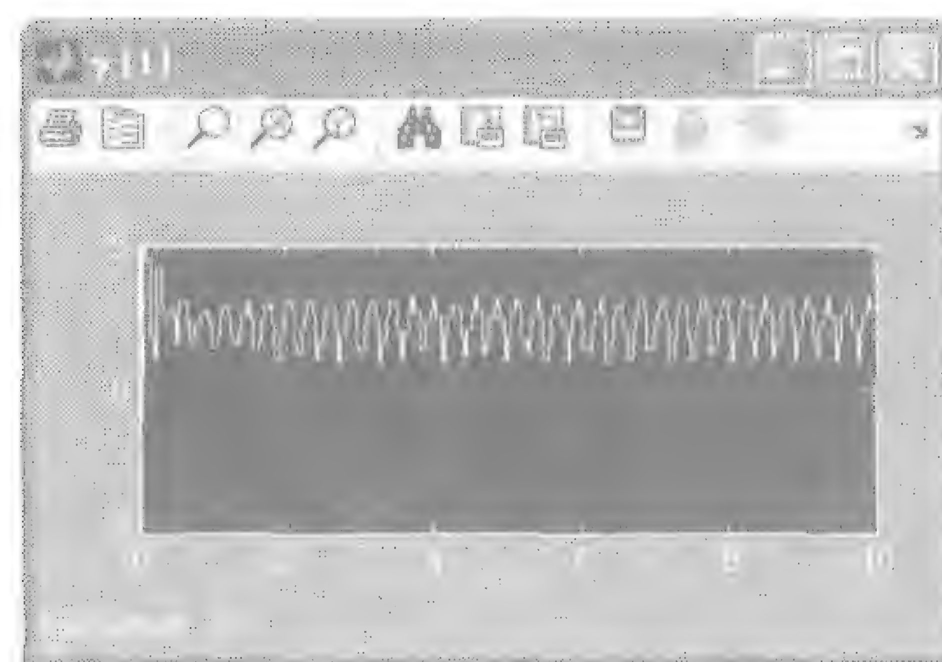


图 8-50 系统直接仿真的结果

可以看出，该波形并不能反映峰值检波的过程，这是因为输入向量只有一个值，双击 Input1 模块可以查看其值为 0.95。如果要观察动态检波过程，则需要对系统模型进行修改。首先，在 MATLAB 命令窗口输入 Simulink，打开“Simulink Library Browser”窗口，然后，按照 Simulink 的一般操作方法，修改系统模型，修改完后的系统模型如图 8-51 所示。图中，信号源 s(t)为调制信号，频率为 1rad/s；信号源 c(t)为载波信号，频率为 20rad/s。AM(t)为已调波信号，y(t)为

峰值检波的输出信号。示波器绘出了 s(t)、AM(t)和 y(t)的波形（Wave），如图 8-52 所示。可以看出，其结果与例 6-2 的仿真结果具有相同的效果。

从以上过程可以看出，在 Simulink 环境中对神经网络进行动态仿真的步骤如下。

（1）在命令窗口或以程序完成神经网络的设计与训练；

（2）以 gensim（net,st）函数生成包括神经网络模块在内的动态仿真模型；

（3）以 Simulink 命令打开“Simulink Library Browser”窗口，根据仿真欲达到的目的，按照 Simulink 的一般操作方法，修改系统模型；

（4）以 Simulink 进行仿真，输出动态仿真结果。

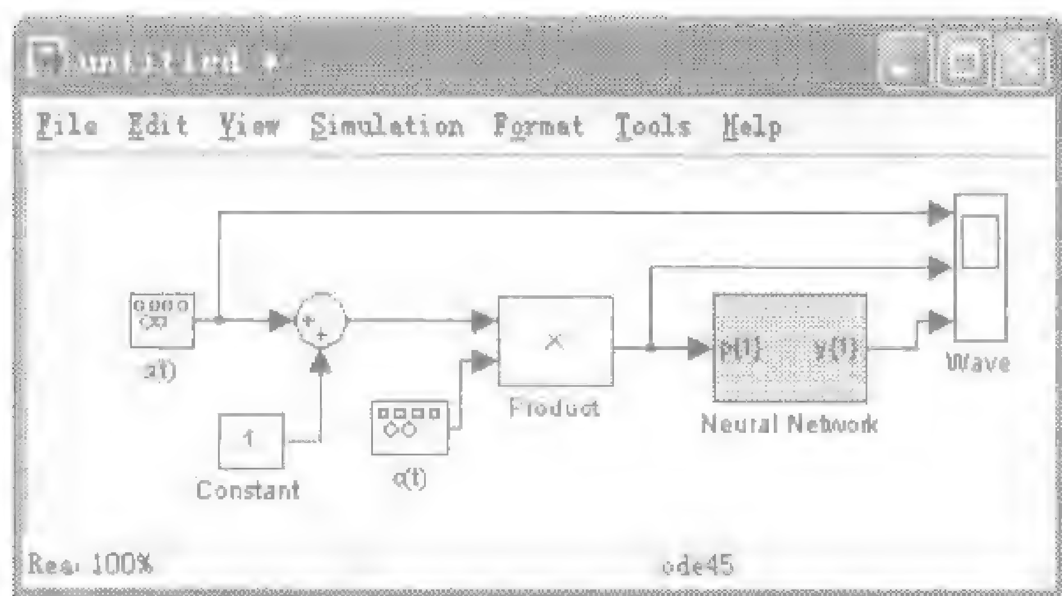


图 8-51　Elman 神经网络峰值检波动态仿真模型

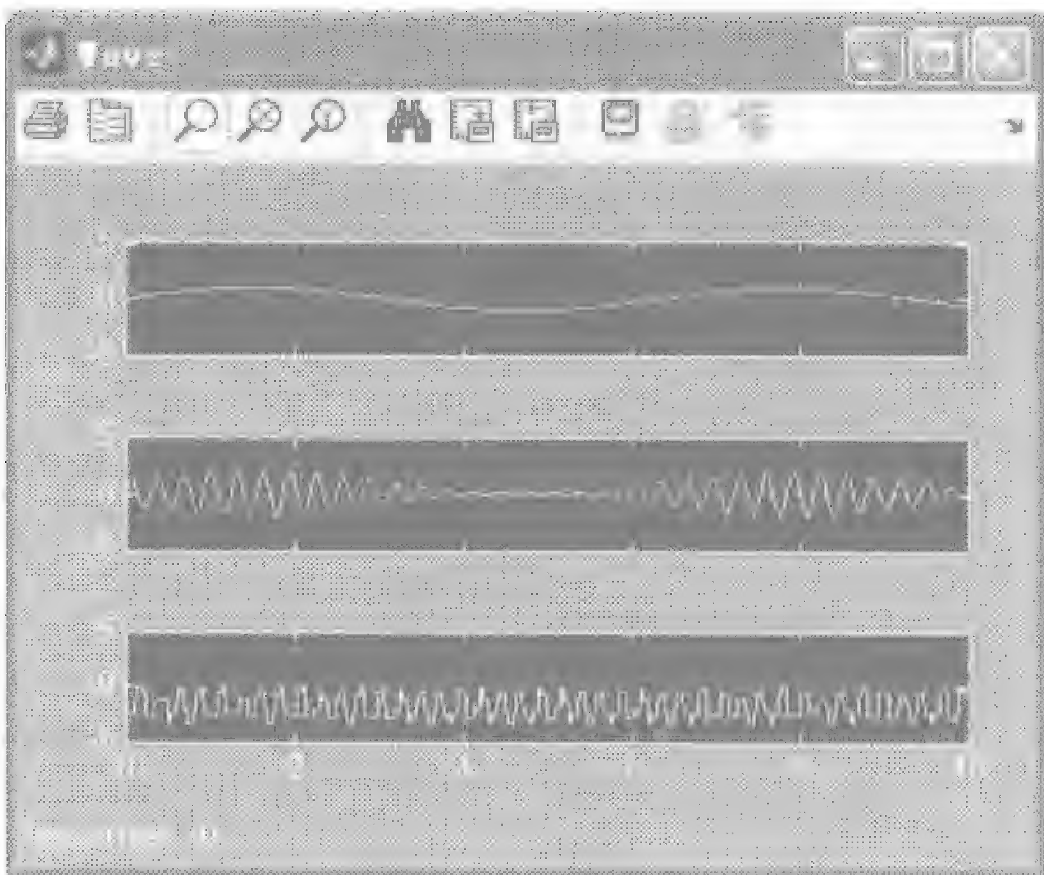


图 8-52　动态仿真动态

第 9 章　自定义神经网络

前面已经详细介绍了 MATLAB 神经网络工具箱中的各种模型以及创建这些网络的标准工具箱函数，就一般应用而言，这些网络模型已足够。可是如果用户欲开发其他更复杂的网络模型，这些标准函数就不能满足用户的需求了，此时，神经网络工具箱提供了另外一种途径，就是用户可以根据需要自定义神经网络。

9.1　自定义神经网络

假设图 9-1 所示网络的输入向量均为输入序列向量

$$\boldsymbol{p}_1=\{[0;0][2;0.5]\},\quad \boldsymbol{p}_2=\{[2;-2;1;0;1][-1;-1;1;0;1]\}$$

希望从第 2 个网络层得到对 $\boldsymbol{p}_1$、$\boldsymbol{p}_2$ 的延迟响应输出($\boldsymbol{y}^1$)；从第 3 个网络层得到目标序列响应输出 $\boldsymbol{y}^2$)，其目标序列向量为

$$\boldsymbol{T}=\{1\quad -1\}$$

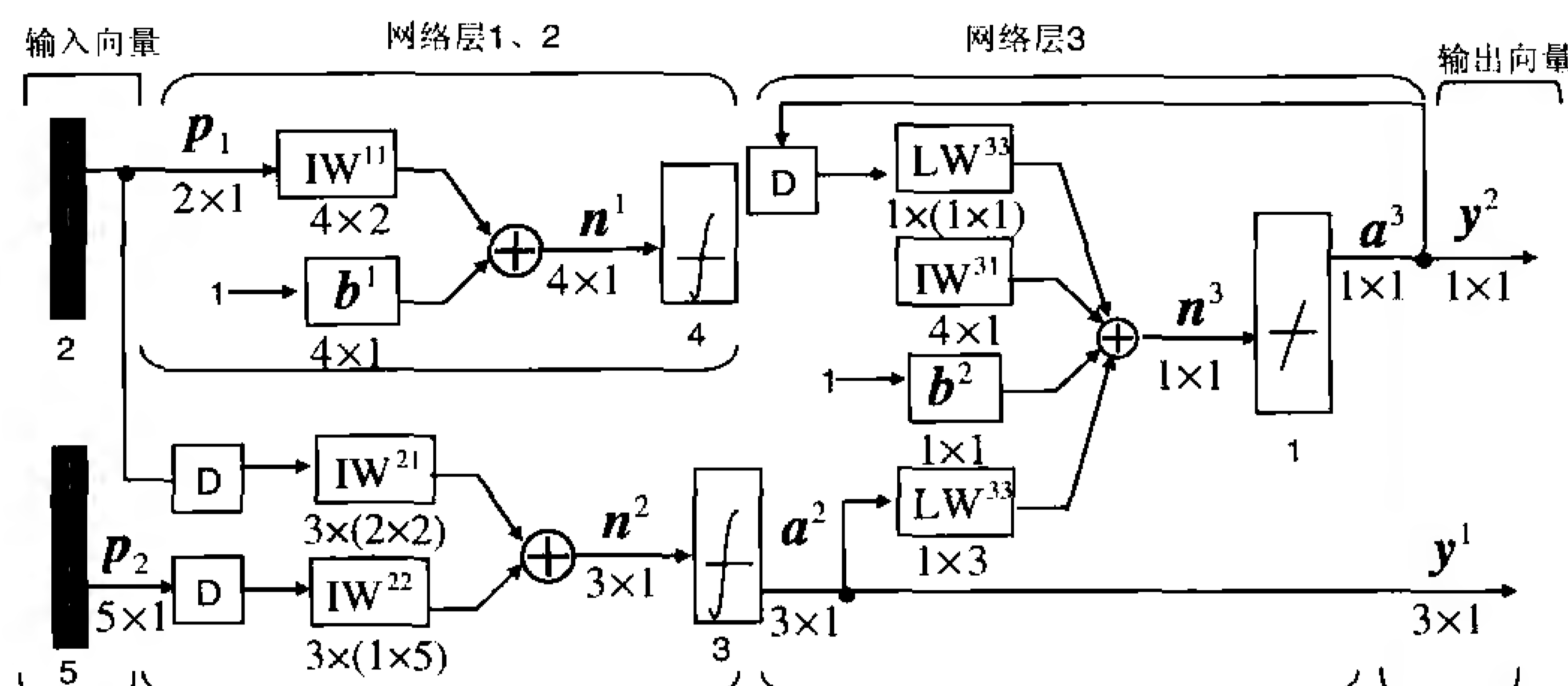


图 9-1　用户自定义网络

由于神经网络工具箱没有提供创建该网络的函数，所以需要自定义网络。下面就以此为例，介绍该用户自定义网络的创建方法。

自定义网络的第一步就是如何通过定义网络对象及其子对象属性创建网络。

9.1.1　自定义神经网络的创建

首先以下列命令生成自定义神经网络的结构。

```
>> net=network
```

其运行结果如下。

```
net =
    Neural Network object:
    architecture:
        numInputs: 0
```

```
          numLayers: 0
        biasConnect: [ ]
       inputConnect: [ ]
       layerConnect: [ ]
      outputConnect: [ ]
      targetConnect: [ ]
         numOutputs: 0  (read-only)
         numTargets: 0  (read-only)
     numInputDelays: 0  (read-only)
     numLayerDelays: 0  (read-only)
   subobject structures:
             inputs: {0×1 cell} of inputs
             layers: {0×1 cell} of layers
            outputs: {1×0 cell} containing no outputs
            targets: {1×0 cell} containing no targets
             biases: {0×1 cell} containing no biases
       inputWeights: {0×0 cell} containing no input weights
       layerWeights: {0×0 cell} containing no layer weights
   functions:
           adaptFcn: (none)
            initFcn: (none)
         performFcn: (none)
           trainFcn: (none)
   parameters:
         adaptParam: (none)
          initParam: (none)
       performParam: (none)
         trainParam: (none)
   weight and bias values:
                 IW: {0×0 cell} containing no input weight matrices
                 LW: {0×0 cell} containing no layer weight matrices
                  b: {0×1 cell} containing no bias vectors
   other:
           userdata: (user stuff)
```

可以看出，所有的属性都为 0 或空值，需要根据需要重新进行设置。对于需要设置的属性，主要有 5 个方面：网络对象结构属性（neural network object architecture）、子对象结构属性（subobject structures：）、函数（functions）、参数（parameters）、权值和阈值（weight and bias values）。当设置或修改其中的任何一个属性值时，与之相关的属性都会自动改变，换句话说，并不是所有的属性都需要设置。按照这几个方面，考察图 9-1 所示的网络。

1．网络结构属性

（1）若网络有 2 个输入向量，则 numInputs=2。

（2）若网络有 3 个网络层，则 numLayers=3。

（3）若网络只有在第 1、3 网络层有阈值向量，第 2 网络层无阈值向量，则 biasConnect= [1 0 1]。

（4）两个输入向量与网络层的连接：第 1、2 网络层有来自输入向量 $\boldsymbol{p}_1$ 的连接；第 2 网络层还有来自输入向量 $\boldsymbol{p}_2$ 的连接；$\boldsymbol{p}_1$、$\boldsymbol{p}_2$ 与第 3 网络层均无连接，故 inputConnect=[1 0; 1 1;0 0]。

（5）网络层之间的连接：第 3 个网络层有来自第 1、2 网络层以及自身的反馈连接，除此之外，无其他网络层的连接，故 layerConnect=[0 0 0 ;0 0 0;1 1 1].

（6）输出向量及其与网络层的连接：网络有 2 个输出量，y^1 来自第 2 个网络层，y^2 来自第 3 个网络层，故 outputConnect=[0 1 1]，而 numOutputs=2 由输出向量与网络层的连接关系自动生成，不需要设置。

（7）目标向量及其与网络层的连接：网络唯一的一个目标向量与第 3 个网络层连接，故 targetConnect=[0 0 1]，而 numTargets=1，由目标向量与网络层的连接关系自动生成，不需要设置。

（8）输入和输出延迟量：在设置了输入层和网络层连接权的延迟后（见“2.子对象结构属性”中的 6）和 7）），可以自动生成网络的输入和输出延迟量，所以在此不必设置。

按照以上分析，以下列命令设置网络对象结构。

```
net.numInputs=2;
net.numLayers=3;
net.biasConnect=[1 0 1]';
net.inputConnect=[1 0;1 1;0 0];
net.layerConnect=[0 0 0;0 0 0;1 1 1];
net.outputConnect=[0 1 1];
net.targetConnect=[0 0 1];
```

由此得到网络结构属性为

```
net =
    Neural Network object:
    architecture:
         numInputs: 2
         numLayers: 3
       biasConnect: [1; 0; 1]
      inputConnect: [1 0; 1 1; 0 0]
      layerConnect: [0 0 0; 0 0 0; 1 1 1]
     outputConnect: [0 1 1]
     targetConnect: [0 0 1]
        numOutputs: 2   (read-only)
        numTargets: 1   (read-only)
    numInputDelays: 0   (read-only)
    numLayerDelays: 0   (read-only)
……
```

2. 子对象结构属性

子对象结构在设置了网络结构属性后会自动生成。

```
net =
    Neural Network object:
……
subobject structures:
             inputs: {2×1 cell} of inputs
             layers: {3×1 cell} of layers
            outputs: {1×3 cell} containing 2 outputs
            targets: {1×3 cell} containing 1 target
             biases: {3×1 cell} containing 2 biases
```

```
    inputWeights: {3×2 cell} containing 3 input weights
    layerWeights: {3×3 cell} containing 3 layer weights
......
```

但其属性值需要根据自定义网络进行修改或重新设置。

1）输入向量

$\boldsymbol{P}_1$ 有 2 个输入元素，其取值范围为 0~2；P_2 有 5 个输入元素，其取值范围为-2~2，只需设置输入向量的取值范围，即

```
net.inputs{1}.range=[0 2;0 2]
net.inputs{2}.range=[-2 2;-2 2;-2 2;-2 2;-2 2]
```

输入向量的 size 属性会自动设置。输入上面的设置后，再输入 net.inputs{1}，net.inputs{2}得到

```
>> net.inputs{1}
ans =
          range: [2×2 double]
           size: 2
       userdata: [1×1 struct]
>> net.inputs{2}
ans =
          range: [5×2 double]
           size: 5
       userdata: [1x1 struct]
```

2）网络层

网络层在设置了网络结构和子对象结构属性后会自动生成，例如对第 1 个网络层而言，有

```
>> net.layers{1}
ans =
     dimensions: 1
    distanceFcn: ''
      distances: []
        initFcn: 'initwb'
    netInputFcn: 'netsum'
      positions: 0
           size: 1
    topologyFcn: 'hextop'
    transferFcn: 'purelin'
       userdata: [1×1 struct]
```

可以看出，其属性值需要根据自定义网络进行修改或重新设置。第 1 层有 4 个神经元；第 2 层有 3 个神经元；而第 3 层的神经元由其输出向量决定，只有 1 个神经元。故只需设置第 1、2 层的神经元数，net.layers{1}.size=4，net.layers{2}.size=3；选择 initFcn='initnw'；第 1 层的传输函数为 transferFcn='tansig'，第 2 层的传输函数为 transferFcn='logsig'。输入以下命令进行网络层设置。

```
net.layers{1}.size=4;
net.layers{1}.initFcn='initnw';
net.layers{1}.transferFcn='tansig';
net.layers{2}.size=3;
```

```
net.layers{2}.initFcn='initnw';
net.layers{2}.transferFcn='logsig';
net.layers{3}.initFcn='initnw'
```

由此得到如下各网络层的属性。

```
>> net.layers{1}
ans =
     dimensions: 4
    distanceFcn: ''
      distances: [ ]
        initFcn: 'initnw'
    netInputFcn: 'netsum'
      positions: [0 1 2 3]
           size: 4
    topologyFcn: 'hextop'
    transferFcn: 'tansig'
       userdata: [1×1 struct]
>> net.layers{2}
ans =
     dimensions: 3
    distanceFcn: ''
      distances: [ ]
        initFcn: 'initnw'
    netInputFcn: 'netsum'
      positions: [0 1 2]
           size: 3
    topologyFcn: 'hextop'
    transferFcn: 'logsig'
       userdata: [1×1 struct]
>> net.layers{3}
ans =
     dimensions: 1
    distanceFcn: ''
      distances: [ ]
        initFcn: 'initnw'
    netInputFcn: 'netsum'
      positions: 0
           size: 1
    topologyFcn: 'hextop'
    transferFcn: 'purelin'
       userdata: [1×1 struct]
```

3）输出向量

在定义网络结构时自动生成输出向量属性如下。

```
>> net.outputs{1}
ans =
     [ ]
>> net.outputs{2}
```

```
ans =
          size: 3
      userdata: [1×1 struct]
>> net.outputs{3}
ans =
          size: 1
      userdata: [1×1 struct]
```

4）目标向量

在定义网络结构时自动生成的目标向量属性如下。

```
>> net.targets{1}
ans =
     [ ]
>> net.targets{2}
ans =
     [ ]
>> net.targets{3}
ans =
          size: 1
      userdata: [1×1 struct]
```

5）阈值向量

在定义网络结构时自动生成的阈值向量属性如下。

```
>> net.biases{1}
ans =
       initFcn: ''
         learn: 1
      learnFcn: ''
    learnParam: ''
          size: 4
      userdata: [1×1 struct]
>> net.biases{2}
ans =
     [ ]
>> net.biases{3}
ans =
       initFcn: ''
         learn: 1
      learnFcn: ''
    learnParam: ''
          size: 1
      userdata: [1×1 struct]
```

6）输入权值向量

在定义网络结构时自动生成输入权值向量属性如下。

```
>> net.inputWeights{1,1}
ans =
        delays: 0
```

```
        initFcn: ''
          learn: 1
        learnFcn: ''
      learnParam: ''
           size: [4 2]
        userdata: [1×1 struct]
       weightFcn: 'dotprod'
>> net.inputWeights{1,2}
ans =
     [ ]
>> net.inputWeights{2,1}
ans =
          delays: 0
        initFcn: ''
          learn: 1
        learnFcn: ''
      learnParam: ''
           size: [3 2]
        userdata: [1×1 struct]
       weightFcn: 'dotprod'
>> net.inputWeights{2,2}
ans =
          delays: 0
        initFcn: ''
          learn: 1
        learnFcn: ''
      learnParam: ''
           size: [3 5]
        userdata: [1×1 struct]
       weightFcn: 'dotprod'
>> net.inputWeights{3,1}
ans =
     [ ]
>> net.inputWeights{3,2}
ans =
     [ ]
```

根据图 9-1 所示，第 1、2 网络层与输入向量的连接权有延迟，net.inputWeights{2,1}. delays=[0 1]，net.inputWeights{2,2}.delays=1，第 3 网络层与自身输出的连接权也有延迟，net.layerWeights{3,3}.delays=1。输入下列命令重新设置。

```
net.inputWeights{2,1}.delays=[0 1];
net.inputWeights{2,1}.delays=1;
```

从结果可以看出，除了设置的延迟外，网络结构中的 numInputDelays 的值由 0 变成了 1。

7）网络层权值向量

在定义网络结构时自动生成的网络层权值向量属性如下。

```
>> net.layerWeights{1,1}
```

```
ans =
        [ ]
>> net.layerWeights{1,2}
ans =
        [ ]
>> net.layerWeights{1,3}
ans =
        [ ]
>> net.layerWeights{2,1}
ans =
        [ ]
>> net.layerWeights{2,2}
ans =
        [ ]
>> net.layerWeights{2,3}
ans =
        [ ]
>> net.layerWeights{3,1}
ans =
            delays: 0
           initFcn: ''
             learn: 1
          learnFcn: ''
        learnParam: ''
              size: [1 4]
          userdata: [1×1 struct]
         weightFcn: 'dotprod'
>> net.layerWeights{3,2}
ans =
            delays: 0
           initFcn: ''
             learn: 1
          learnFcn: ''
        learnParam: ''
              size: [1 3]
          userdata: [1×1 struct]
         weightFcn: 'dotprod'
>> net.layerWeights{3,3}
ans =
            delays: 0
           initFcn: ''
             learn: 1
          learnFcn: ''
        learnParam: ''
              size: [1 1]
          userdata: [1×1 struct]
         weightFcn: 'dotprod'
```

根据图 9-1 所示，第 3 网络层与自身输出的连接权也有延迟，net.layerWeights{3,3}. delays=1。输入下列命令重新设置。

```
net.layerWeights{3,3}.delays=1;
```

从结果可以看出，除了设置的延迟外，网络结构中的 numLayerDelays 的值由 0 变成了 1。

3. 函数属性

以下列命令设置函数属性。

```
net.initFcn='initlay';
net.performFcn='mse';
net.trainFcn='trainlm';
```

输入 net，观察函数属性结果为

```
……
functions:
          adaptFcn: (none)
           initFcn: 'initlay'
        performFcn: 'mse'
          trainFcn: 'trainlm'
……
```

4. 参数属性

当确定了函数属性后，函数的参数属性以各函数的默认值自动生成。输入 net，观察函数属性结果为

```
……
parameters:
        adaptParam: (none)
         initParam: (none)
      performParam: (none)
        trainParam: .epochs, .goal, .max_fail, .mem_reduc,
                    .min_grad, .mu, .mu_dec, .mu_inc,
                    .mu_max, .show, .time
……
```

5. 权值和阈值属性

当网络结构属性和子对象属性确定下来以后，网络权值和阈值的结构就确定了。

```
>> net.iw
ans =
    [4×2 double]                  []
    [3×2 double]     [3×5 double]
              []                  []
>> net.lw
ans =
              []           []           []
              []           []           []
    [1×4 double]   [1×3 double]        [0]
>> net.b
ans =
```

```
    [4×1 double]
              []
    [         0]
```

由于网络还未进行初始化，因此所有权值和阈值的具体数据全为 0。

至此，完成了对图 9-1 所示网络的创建工作，键入 net，可以查看所定义的网络相关属性。

```
>> net
net =
    Neural Network object:
    architecture:
         numInputs: 2
         numLayers: 3
       biasConnect: [1; 0; 1]
      inputConnect: [1 0; 1 1; 0 0]
      layerConnect: [0 0 0; 0 0 0; 1 1 1]
     outputConnect: [0 1 1]
     targetConnect: [0 0 1]
        numOutputs: 2  (read-only)
        numTargets: 1  (read-only)
    numInputDelays: 1  (read-only)
    numLayerDelays: 1  (read-only)
    subobject structures:
            inputs: {2×1 cell} of inputs
            layers: {3×1 cell} of layers
           outputs: {1×3 cell} containing 2 outputs
           targets: {1×3 cell} containing 1 target
            biases: {3×1 cell} containing 2 biases
      inputWeights: {3×2 cell} containing 3 input weights
      layerWeights: {3×3 cell} containing 3 layer weights
    functions:
          adaptFcn: (none)
           initFcn: 'initlay'
        performFcn: 'mse'
          trainFcn: 'trainlm'
    parameters:
        adaptParam: (none)
         initParam: (none)
      performParam: (none)
        trainParam: .epochs, .goal, .max_fail, .mem_reduc,
                    .min_grad, .mu, .mu_dec, .mu_inc,
                    .mu_max, .show, .time
    weight and bias values:
                IW: {3×2 cell} containing 3 input weight matrices
                LW: {3×3 cell} containing 3 layer weight matrices
                 b: {3×1 cell} containing 2 bias vectors
    other:
          userdata: (user stuff)
```

9.1.2 自定义神经网络的初始化、训练与仿真

1. 自定义神经网络的初始化

自定义网络可以调用 init 函数，以定义的权值和阈值初始化函数对网络的权值和阈值进行初始化。

```
>> net=init(net);
```

初始化后的权值和阈值已经不再全部为 0，其结果为

```
>> net.iw
ans =
     [4×2 double]              []
     [3×2 double]     [3×5 double]
               []              []
>> net.iw{1,1}
ans =
   -2.3908     1.4575
    2.2937    -1.6059
    1.9985     1.9611
   -2.8000     0.0164
>> net.iw{2,1}
ans =
    0.4189    -0.6207
   -0.1422    -0.6131
   -0.3908     0.3644
>> net.iw{2,2}
ans =
   -0.3945     0.3958     0.7073     0.7995     0.6359
    0.0833    -0.2433     0.1871     0.6433     0.3205
   -0.6983     0.7200    -0.0069     0.2898    -0.3161
>> net.lw
ans =
                []              []              []
                []              []              []
     [1x4 double]    [1×3 double]       [-0.2592]
>> net.lw{3,1}
ans =
   -0.4205    -0.3176     0.0682     0.4542
>> net.lw{3,2}
ans =
   -0.3814     0.6770     0.1361
>> net.lw{3,3}
ans =
   -0.2592
>> net.b
ans =
     [4x1 double]
               []
```

```
    [      0.4055]
>> net.b{1}
ans =
     3.7333
    -1.6211
    -3.0263
    -0.0164
>> net.b{2}
ans =
      []
>> net.b{3}
ans =
     0.4055
```

2. 自定义神经网络的训练

自定义网络可以调用 train 函数，以定义的网络训练函数对网络进行训练，由于第 3 个网络层定义了目标向量，所以训练前必须定义输入向量和目标向量。

```
P={[0;0] [2;0.5];[2;-2;1;0;1] [-1;-1;1;0;1]};
T={1 -1};
net=train(net,P,T);
```

训练结果为

```
TRAINLM, Epoch 0/100, MSE 0.498415/0, Gradient 2.50773/1e-010
TRAINLM, Epoch 5/100, MSE 6.40949e-031/0, Gradient 2.9101e-015/1e-010
TRAINLM, Minimum gradient reached, performance goal was not met.
```

训练的误差性能曲线如图 9-2 所示。

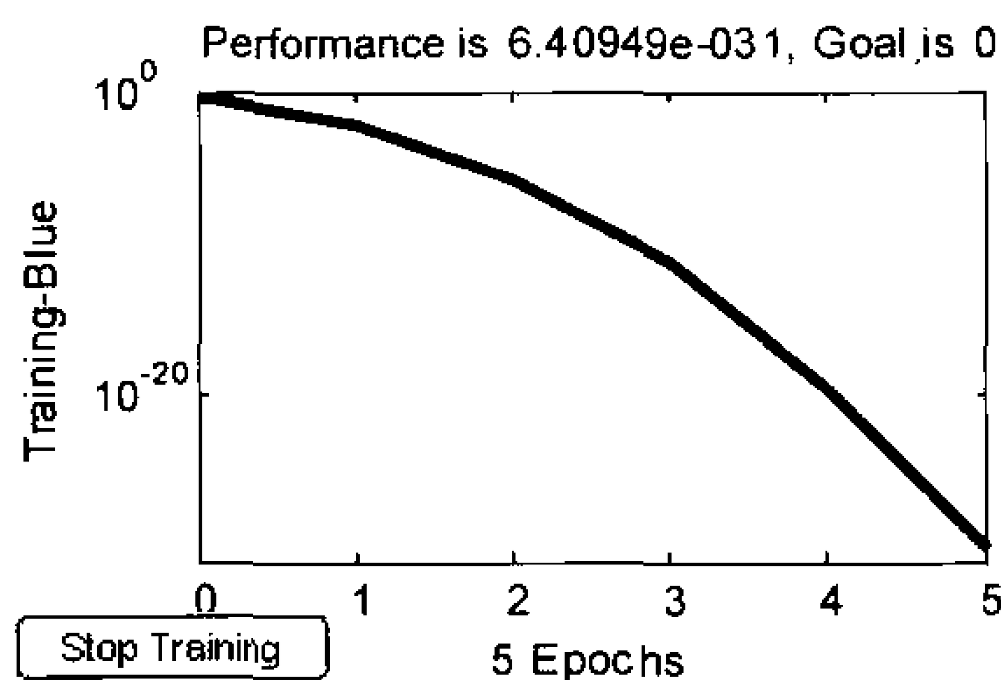


图 9-2 训练误差性能曲线

3. 自定义神经网络的仿真

自定义网络可以调用 sim 函数对网络进行仿真，若以训练样本进行仿真，则

```
P={[0;0] [2;0.5];[2;-2;1;0;1] [-1;-1;1;0;1]};
y=sim(net,P)
```

仿真结果为

```
>> y=sim(net,P)
y =
    [3×1 double]      [3×1 double]
    [     1.0000]     [    -1.0000]
>> y{1,1}
```

```
ans =
      0.4260
      0.7727
      0.0406
>> y{1,2}
ans =
      0.7951
      0.6491
      0.4108
```

第 1 个网络输出为 y{1,1}和 y{1,2}；第 2 个网络输出（第 3 个网络层的输出）如下。

```
>> y2={y{2,1},y{2,2}}
y2 =
      [1.0000]      [-1.0000]
```

可以看出，与目标向量是一致的。

9.2　自定义函数

神经网络工具箱允许创建并应用很多种类的函数，用户可以根据需要，在初始化、仿真及训练中应用多种方法，实现对网络的自行调整。这些自行编制的函数称为自定义函数。

本节介绍如何创建自定义函数，这些函数主要用来进行神经网络的初始化、学习、训练和仿真，可以大致分为四类。

1）初始化函数

- 网络初始化函数
- 层初始化函数
- 权值和阈值初始化函数

2）学习函数

- 网络训练函数
- 网络自适应函数
- 网络性能函数
- 权值和阈值学习函数

3）仿真函数

- 传递函数
- 传递函数导函数
- 网络输入函数
- 网络输入函数导函数
- 权值函数
- 权值导函数

4）自组织映射函数

- 拓扑函数
- 距离函数

9.2.1 初始化函数

初始化函数包括网络、层、权值和阈值三类初始化函数，下面分别介绍如何自定义生成这三种函数。

1．网络初始化函数

网络初始化函数将所有的权值和阈值设置为一个适当的值，作为网络训练或者自适应调节的初始点。一旦定义了网络初始化函数，就可以嵌入到某一网络中。假设定义网络初始化函数为 xiu()，并嵌入某一网络中，则应用下面语句实现。

net.initFcn='xiu'

这样，在调用 init() 初始化网络时，都可以应用此时设定的网络初始化函数进行初始化。

net=init (net)

网络初始化函数编制完成后，接受某一网络，并且在初始化处理以后，再返回一个网络。

net=xiu (net, i)

自定义网络初始化函数可以根据要求对权值和阈值进行任意设置。

2．层初始化函数

层初始化函数将某层所有的权值和阈值设置为一个适当的值，作为网络训练或者自适应调节的初始点。一旦定义了层初始化函数，就可以嵌入到网络任意一层上。假设定义的层初始化函数为 xiu()，并嵌入网络第 3 层上，则应用下面语句实现。

net.layers{3}.initFcn='xiu'

如果网络初始化函数（net.initFcn）设置为工具箱函数 initlay()，自定义层初始化函数用来对层进行初始。那么在调用 init() 初始化网络时，都可以应用此时设定的层初始化函数进行初始化。

net=init (net)

层初始化函数编制完成后，接受网络和层的标号作为输入 i，并且在对第 i 层初始化处理以后，再返回网络。

net=xiu (net, i)

自定义层初始化函数可以根据要求对层的权值和阈值进行任意设置。

3．权值和阈值初始化函数

权值和阈值初始化函数将所有的权值和阈值设置为一个适当的值，作为网络训练或者自适应调节的初始点。一旦定义了权值和阈值初始化函数，就可以嵌入到网络任意权值和阈值上。假设定义了权值和阈值初始化函数为 xium()，并嵌入网络第 2 层的阈值、第 1 层输入到第 2 层的权值上，则应用下面的语句实现。

net=init (net)

权值和阈值初始化函数编制完成后，可以应用如下方式进行调用。

W=rands (S, PR)

b=rands (S)

其中，S 为层神经元数目；PR 为 R 个输入向量的最大/最小值矩阵。

神经网络工具箱中包含一个自定义权值和阈值初始化函数 mywbif()，输入 help myxiu 就可以获得有关此函数的帮助信息。举例说明如何应用 myxiu()进行权值和阈值的初始化。

【例 9-1】

```
w=mywbif(4,[0 2;-2 3])
b=mywbif(4,[1 1])
```

输出结果为

```
w =
     0.0173      0.0876
     0.0980      0.0737
     0.0271      0.0137
     0.0252      0.0012
b =
     0.0894
     0.0199
     0.0299
     0.0661
```

输入 type mywbif，可以查看 mywbif()的源程序。

```
>> type mywbif
function w = mywbif(s,pr)
%MYWBIF Example custom weight and bias initialization function.
%
%   Use this function as a template to write your own function.
%
%   Syntax
%
%      W = rands(S,PR)
%         S- number of neurons.
%         PR - Rx2 matrix of R input ranges.
%         W - SxR weight matrix.
%
%      b = rands(S)
%         S- number of neurons.
%         b - Sx1 bias vector.
%
%   Example
%
%      W = mywbif(4,[0 1; -2 2])
%      b = mywbif(4,[1 1])
% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.2.2.1 $

if nargin < 1, error('Not enough input arguments'), end
if nargin == 1
   w = rand(s,1)*0.2;   % <-- Replace with your own initial bias vector
```

```
else
  r = size(pr,1);        % <-- Replace with your own initial weight matrix
  w = rand(s,r)*0.1;
end
```

可以将函数 mywbif()作为一个模板，用来生成自定义的权值和阈值初始化函数。

9.2.2 学习函数

与网络学习、权值和阈值调整有关系的学习函数有四类：训练函数、自适应函数、性能函数、权值和阈值调整函数，下面分别介绍如何自定义生成这些函数。

1. 训练函数

训练函数是常用的学习函数。学习函数循环地将输入向量应用于网络中，每次都能够更新网络，直到达到训练目标位置。训练停止的条件可以是最大学习次数、最小的误差梯度或者训练精度等。一旦定义了训练函数，就可以嵌入到某一网络上。假设定义了训练函数为 xiu()，并嵌入某个网络中，则应用下面语句实现。

net.trainFcn='xiu'

这样，训练网络时，都可以应用此时设定的训练函数。

[net, tr]=train (NET, P, T, Pi, Ai)

训练函数编制完成后，可以应用如下方式进行调用。

[net, tr]=xiu (net, Pd, T1, Ai, Q, TS, VV, TV)

自定义训练函数需要提供如下信息。

- version：神经网络工具箱的版本。
- pdefaults：默认参数。

自定义训练函数可以根据所设定的任意方式更新网络的权值和阈值。

2. 自适应函数

自适应函数在每个输入时间段内都要更新网络，并进行仿真。一旦定义了自适应函数，就可以嵌入到某一网络上。假设定义的自适应函数为 xium()，并嵌入了某个网络中，则应用下面语句实现。

net.adaptFcn='xium';

这样，自适应调节网络时，都可以应用此时设定的自适应函数。

[net, Y, E, Pf, Af]=adapt (NET, P, T, Pi, Ai)

自适应函数编制完成后，可以应用如下方式进行调用：

[net, Ac, E1]=xium (net, Pd, Ti, Ai, Q, TS)

自定义自适应函数需要提供如下信息。

- version：神经网络工具箱的版本。
- pdefaults：默认参数。

自定义自适应函数可以根据所设定的任意方式更新网络的权值和阈值。

3. 性能函数

性能函数是学习训练时期望达到最优的指标，通过改变权值和阈值使性能函数最优，改善

网络性能。一旦定义了性能函数，就可以嵌入到某一网络上。假设定义性能函数为 xiuw()，并嵌入某个网络中，则应用下面语句实现。

net.performFcn='xiuw'

这样，训练或者自适应调节网络时，都可以应用此时设定的性能函数进行优化。

[net, tr]=train (NET, P, T, Pi, Ai)

[net, Y, E, Pf, Af]=adapt (NET, P, T, Pi, Ai)

性能函数编制完成后，可以应用如下方式进行调用。

perf=xiuw (E, X, PP)

其中，E 为期望值矩阵；X 为网络权值和阈值；PP 为网络参数。

如果 E 是细胞数组，则需要首先转换成矩阵形式，然后再进行调用，X 和 PP 的值通过网络得到。

```
E=cell2mat (E);
perf=xiuw (E, net)
X=getx (net);
PP=net.performParam;
```

自定义性能函数需要提供如下信息。

- version：神经网络工具箱的版本。
- deriv：相关导数函数名。
- pdefaults：默认参数。

神经网络工具箱中包含一个自定义性能函数 mypf()，输入 help mypf 就可以获得有关此函数的帮助信息。举例说明如何应用 mypf() 性能函数，随机输入网络权值和阈值，并应用网络默认参数，计算相应的性能函数值。

【例 9-2】

```
e=rand(4,5);
x=rand(12,1);
pp=mypf('pdefaults')
perf=mypf(e,x,pp)
```

输入结果为

```
pp =
    x: 1
    y: 0.5000
perf =
    0.7721
```

输入 type mypf，可以查看 mypf() 的源程序。

```
>> type mypf
function perf = mypf(e,x,pp)
%MYPF Example custom performance function.
%
%   Use this function as a template to write your own function.
```

```
%
% Calculation Syntax
%
%    perf = mypf(E,X,PP)
%       E   - Matrix or cell array of error vector(s).
%       X   - Vector of all weight and bias values.
%       PP - Performance parameter.
%
%    perf = mypf(E,net)
%
% Information Syntax
%
%    info = mytf(code) returns useful information for each CODE string:
%       'version' - Returns the Neural Network Toolbox version (3.0).
%       'deriv'    - Returns the name of the associated derivative function.
%       'output'   - Returns the output range.
%       'active'   - Returns the active input range.
%
% Example
%
%    e = rand(4,5);
%    x = rand(12,1);
%    pp = mypf('pdefaults')
%    perf = mypf(e,x,pp)
% Copyright 1997-2001 The MathWorks, Inc.
% $Revision: 1.4.2.1 $
if nargin < 1, error('Not enough arguments.'); end
if isstr(e)
   switch (e)
   case 'version'
      perf = 3.0;            % <-- Must be 3.0.
   case 'deriv',
      perf = 'mydpf';        % <-- Replace with the name of your derivative function or ''
   case 'name',
      perf = 'Custom';       % <-- Replace with your function's name
   case 'pnames',
      perf = {};             % <-- Add names of your function parameters (if any)
   case 'pdefaults'
      perf.x = 1;            % <-- Replace with the your own performance
      perf.y = 0.5;          % <-- parameter structure or null matrix [ ].
   otherwise, error('Unrecognized code.')
   end
else
   if isa(e,'cell')
      e = cell2mat(e);
   end
```

```
    if nargin == 2
        pp = x.performParam;          % <-- delete this line if you don't use PP
        x = getx(net);                % <-- delete this line if you don't use X
    end
    % ** Replace the following calculation with your own
    % ** measure of performance.
    numErrors = prod(size(e));
    numWeightsBiases = length(x);
    perf =    sum(sum(abs(e)))* pp.x/numErrors + ...
        sum(abs(x))*pp.y/numWeightsBiases;
end
```

4．权值和阈值学习函数

权值和阈值学习函数与某些训练或者自适应调节函数一起使用，用于学习训练中更新权值和阈值。一旦定义了权值和阈值学习函数，就可以嵌入到网络任意一层上。假设定义的权值和阈值学习函数为 xiu()，并嵌入网络第 2 层，则应用下面语句实现。

```
net.layers{2}.learnFcn='xiu';
```

这样，如果网络训练函数（net.trainFcn）设置为 trainb、trainc 或者 trainr，网络自适应调节函数（net.adaptFcn）设置为 trains，都可以应用此时设定的权值和阈值学习函数更新权值和阈值。

```
[net, tr]=train (NET, P, T, Pi, Ai)
[net, Y, E, Pf, Af]=adapt (NET, P, T, Pi, Ai)
```

权值和阈值学习函数编制完成后，可以应用如下的方式进行调用，以更新权值和阈值。

```
[dW, LS]=xiu (W, P, Z, N, A, T, E, gW, gA, D, LP, LS)
[db, LS]=xiu (b, ones (1, Q), Z, N, A, T, E, gW, gA, D, LP, LS)
```

自定义权值和阈值学习函数需要提供如下信息：

- version：神经网络工具箱的版本。
- deriv：相关导数函数名。
- pdefaults：默认参数。

神经网络工具箱中包含一个自定义权值和阈值学习函数 mywblf ()，输入 help mywblf 就可以获得有关此函数的帮助信息。

```
>> type mywblf
function [dw,ls] = mywblf(w,p,z,n,a,t,e,gW,gA,d,lp,ls)
%MYWBLF Example custom weight and bias learning function.
%
%   Calculation Syntax
%
%       [dW,LS] = mywblf(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
%       [db,LS] = mywblf(b,ones(1,Q),Z,N,A,T,E,gW,gA,D,LP,LS)
%           W   - S×R weight matrix (or Sx1 bias vector).
%           P   - R×Q input vectors (or ones(1,Q)).
%           Z   - S×Q weighted input vectors.
```

```
%         N   - S×Q net input vectors.
%         A   - S×Q output vectors.
%         T   - S×Q layer target vectors.
%         E   - S×Q layer error vectors.
%         gW - S×R gradient with respect to performance.
%         gA - S×Q output gradient with respect to performance.
%         D   - S×S neuron distances.
%         LP - Learning parameters, none, LP = [].
%         LS - Learning state, initially should be = [].
%         dW - S×R weight (or bias) change matrix.
%
%   Information Syntax
%
%      info = mywblf(code) returns useful information for each CODE string:
%         'version'     - Returns the Neural Network Toolbox version (3.0).
%         'pdefaults' - Returns the name of the associated derivative function.
%         'needg'         - Returns the output range.
%
%   Example
%
%      W = rand(4,5);
%      gW = rand(4,5);
%      lp = mywblf('pdefaults')
%      [dW,ls] = mywblf(w,[],[],[],[],[],[],gW,[],[],lp,[]);
%      W = W + dW;
%      gW = rand(4,5);
%      [dW,ls] = mywblf(w,[],[],[],[],[],[],gW,[],[],lp,ls);
%      W = W + dW;
% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.3.2.1 $
if isstr(w)
   switch lower(w)
      case 'version'
      dw = 3.0;                % <-- Must be 3.0.

   case 'pdefaults'
      dw.lr = 0.01;            % <-- Replace with your own learning
                               %         parameters or the null matrix [].
   case 'needg'
      dw = 1;                  % <-- 1 or 0 depending on whether your
                               %         function uses gW or gA, or not.
   otherwise
      error('Unrecognized property.')
   end
else
   if isempty(ls)
```

```
    ls.x = 0.3;          % <-- Replace with your own functions initial
                         %       learning state or the null matrix []
  end
  dw = lp.lr*ls.x*gW;    % <-- Replace with your own weight change
                         %       calculation.
    ls.x = 1-ls.x;       % <-- Replace with your own learning state
                         %       update code, if you have any such state.
end
```

9.2.3 仿真函数

自定义仿真函数包括传递函数、网络输入函数、权值函数。下面分别介绍如何创建这三种仿真函数及其相应的导数函数。

1．传递函数

传递函数根据给定的网络输入向量（或者矩阵）***N***，计算某层的输出向量（或矩阵）***A***，网络输入向量和输出向量必须具有相同的维数。一旦定义了传递函数，就可以嵌入到网络中任意一层上。假设定义了传递函数为 xiux()，并嵌入网络第 3 层上，则应用下面语句实现。

```
net.layers{3}.transferFcn='xiux'
```

这样，在对网络进行仿真时，都可以应用此时设定的传递函数。

```
[Y, Pf, Af]=sim (net, P, Pi, Ai)
```

传递函数编制完成后，可以应用如下方式进行调用。

```
A=xiux (N)
```

其中，N 为网络输入向量；A 为函数返回值，即网络输出向量。

自定义传递函数返回如下信息。

- version：神经网络工具箱的版本。
- deriv：相关导数函数名。
- output：输出范围。
- active：活动的输入范围。

神经网络工具箱中包含了一个自定义传递函数 mytf()，输入 help mytf 就可以获得有关此函数的帮助信息。

【例 9-3】绘制传递函数 mytf() 曲线，如图 9-3 所示。

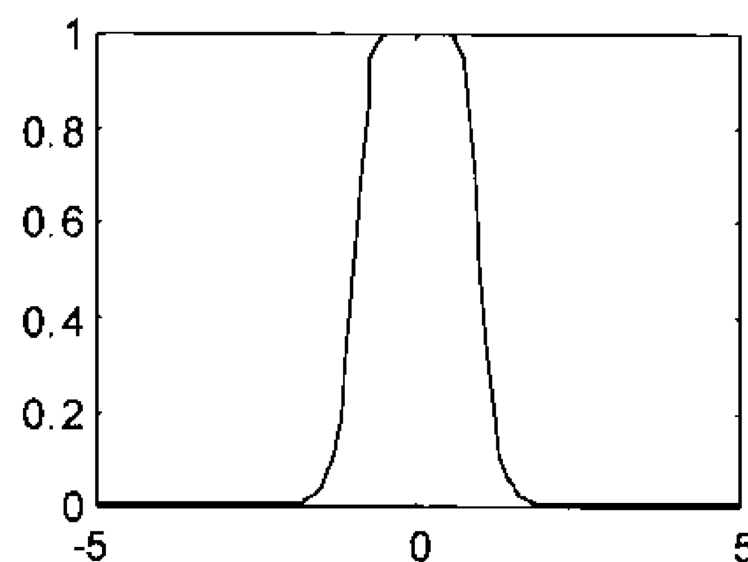


图 9-3　自定义传递函数 mytf ( )曲线

```
N=-5:0.1:5;
A=mytf(N);
plot(N,A)
```

```
mytf('deriv')
ans =
mydtf
```

输入 type mytf，可以查看 mytf()的源程序。

```
>> type mytf
function a = mytf(n)
%MYTF Example custom transfer function.
%
%   Use this function as a template to write your own function.
%
%   Calculation Syntax
%
%     A = mytf(N)
%       N - S×Q matrix of Q net input (column) vectors.
%       A - S×Q matrix of Q output (column) vectors.
%
%   Information Syntax
%
%     info = mytf(code) returns useful information for each CODE string:
%       'version' - Returns the Neural Network Toolbox version (3.0).
%       'deriv'   - Returns the name of the associated derivative function.
%       'output'  - Returns the output range.
%       'active'  - Returns the active input range.
%
%   Example
%
%     n = -5:.1:5;
%     a = mytf(n);
%     plot(n,a)
% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.2.2.1 $
if nargin < 1, error('Not enough arguments.'); end
if isstr(n)
   switch (n)
      case 'version'
      a = 3.0;              % <-- Must be 3.0.
            case 'deriv'
      a = 'mydtf';          % <-- Replace with the name of your
                            %        associated function or ''
      case 'output'
      a = [-1 1];           % <-- Replace with the minimum and maximum
                            %        output values of your transfer function
      case 'active'
      a = [-2 2];           % <-- Replace with the range of inputs where
                            %        the outputs are most sensitive to changes.
```

```
    otherwise, error('Unrecognized code.')
  end
else
  a = 1./(n.^8+1);      % <-- Replace with your calculation
end
```

可以将函数 mytf()作为一个模板，用来生成自定义的传递函数。

2．传递函数导数函数

如果自定义传递函数需要回传，则需要自定义传递函数导数函数，传递函数导数函数根据给定的网络输入来计算此层输出的导数。假设定义传递函数导数函数为 xiudx()，则应用下面语句计算此层输出的导数。

dA_dN=xiudx (N, A)

其中，N 为网络输入向量；A 为网络输出向量；dA_dN 表示导数 dA/dN。

神经网络工具箱中包含了一个自定义传递函数导数函数 mydtf()，输入 help mydtf 就可以获得有关此函数的帮助信息。

【例 9-4】绘制传递函数导数函数曲线，如图 9-4 所示。

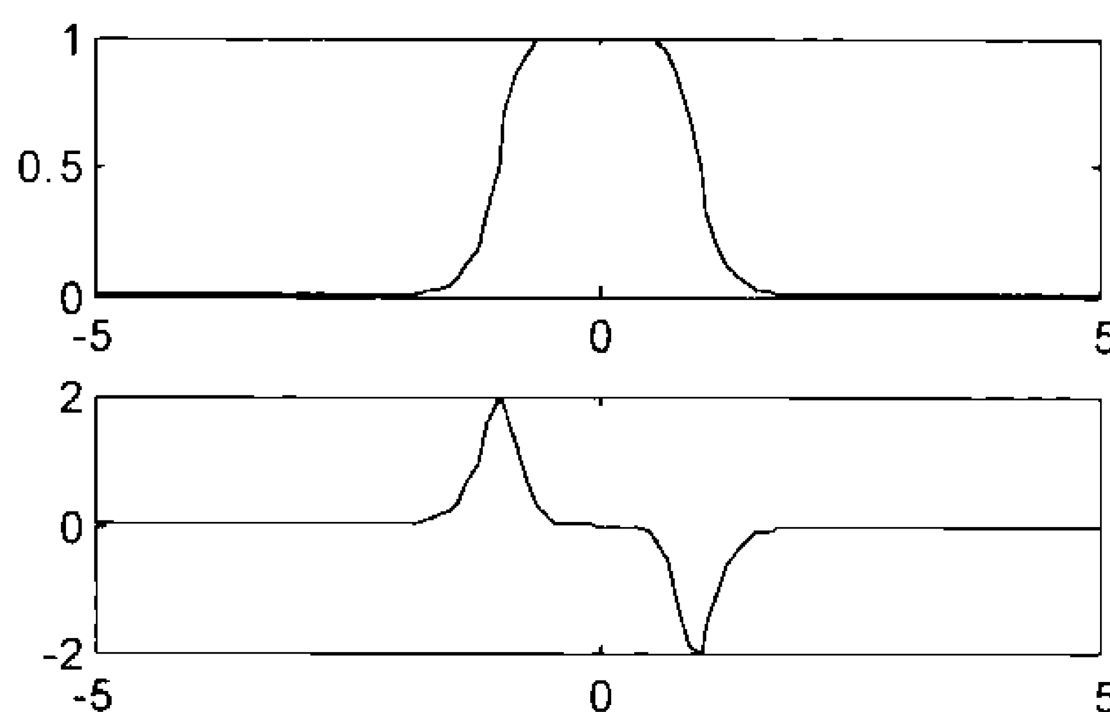


图 9-4　自定义传递函数导数函数 mydtf ()曲线

```
N=-5:0.1:5;
A=mytf(N);
dA_dN=mydtf(N,A);
subplot(211)
plot (N,A)
subplot(212)
plot(N,dA_dN)
```

输入 type mydtf，可以查看 mydtf () 的源程序。

```
>> type mydtf
function d = mydtf(n,a)
%MYDTF Example custom transfer derivative function of MYTF.
%
%   Use this function as a template to write your own function.
%
%   Syntax
%
%     dA_dN = mydtf(N,A)
%       N – S×Q matrix of Q net input (column) vectors.
%       A – S×Q matrix of Q output (column) vectors.
```

```
%         dA_dN – S×Q derivative dA/dN.
%
%   Example
%
%       n = -5:.1:5;
%       a = mytf(n);
%       da_dn = mydtf(n,a);
%       subplot(2,1,1), plot(n,a)
%       subplot(2,1,2), plot(n,da_dn)
% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.2.2.1 $
% **   Replace the following calculation with your
% **   derivative calculation.
d = -8*n.^7. *a.^2;
% **   Note that you have both the transfer functions input N and
% **   output A available, which can often allow a more efficient
% **   calculation of the derivative than with just N.
```

可以将函数 mydtf()作为一个模板，用来生成自定义的传递函数导数函数。

3．网络输入函数

网络输入函数根据给定的加权输入向量（或者矩阵）***Z***，计算某层的网络输入向量（或矩阵）***N***，网络输入向量和加权输入向量必须具有相同的维数。一旦定义了网络输入函数，就可以嵌入到网络中任意一层上。假设定义了网络输入函数为 xiurf()，并嵌入网络第 3 层上，则应用下面语句实现。

net.layers{3}.netInputFcn='xiurf'

这样，在对网络进行仿真时，都可以应用此时设定的网络输入函数。

[Y, Pf, Af]=sim (net, P, Pi, Ai)

网络输入函数编制完成后，可以应用如下方式进行调用。

N=xiurf (Z1, Z2, …)

其中，Z1、Z2……为加权输入向量；N 为函数返回值，即网络输入向量。

自定义网络输入函数返回如下信息。

- version：神经网络工具箱的版本。
- deriv：相关导数函数名。

神经网络工具箱中包含了一个自定义网络输入函数 mynif()，输入 help mynif 就可以获得有关此函数的帮助信息。举例说明如何应用 mynif()进行网络输入向量计算。

【例 9-5】

```
Z1=rand(4,4);
Z2=rand(4,4);
Z3=rand(4,4);
N=mynif(Z1,Z2,Z3)
mynif('deriv')
```

输出结果如下。

```
N =
     0.2289    0.0511    0.1041    0.1678
     0.1541    0.1653    0.0172    0.1194
     0.1605    0.1197    0.1231    0.0138
     0.1797    0.0064    0.1800    0.1630
ans =
deriv
```

输入 type mynif，可以查看 mynif()的源程序。

```
>> type mynif
function n=mynif(varargin)
%MYNIF Example custom net input function.
%
%   Use this function as a template to write your own function.
%
%   Calculation Syntax
%
%      N = mynif(Z1,Z2,...)
%         Zi - S×Q matrix of Q weighted (column) vectors.
%         N - S×Q matrix of Q net input (column) vectors.
%
%   Information Syntax
%
%      info = mynif(code) returns useful information for each CODE string:
%         'version' - Returns the Neural Network Toolbox version (3.0).
%         'deriv'     - Returns the name of the associated derivative function.
%
%   Example
%
%      z1 = rand(4,5);
%      z2 = rand(4,5);
%      z3 = rand(4,5);
%      n = mynif(z1,z2,z3)
% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.2.2.1 $
if nargin < 1, error('Not enough arguments.'); end
n = varargin{1};
if isstr(n)
   switch n
      case 'version'
      a = 3.0;            % <-- Must be 3.0.
      case 'deriv'
      a = 'mydnif';    % <-- Replace with the name of your
                               % associated derivative function or ''
      otherwise
         error('Unrecognized code.')
```

```
    end
else
% **   Replace the following calculation with your own.   The only
% **   constraint is that the function must not be sensitive
% **   to the order of its input arguments.
% **   In other words, MYNIF(Z1,Z2,Z3) must return the same
% **   values as MYNIF(Z2,Z3,Z1), MYNIF(Z1,Z3,Z2), etc.
    n = 1./n;
    for i=2:length(varargin)
        n = n + 1./varargin{i};
    end
    n = 1./n;
end
```

可以将函数 mynif()作为一个模板，用来生成自定义的网络输入函数。

4．网络输入导函数

如果自定义网络输入函数需要回传，则需要自定义网络输入导数函数，网络输入导数函数根据给定的加权输入来计算此层网络输入的导数。假设定义网络输入导数函数为 xiurf()，则应用下面语句计算此层网络输入的导数。

dN_dZ=xiurf (Z, N)

其中，Z 为网络加权输入向量；N 为网络输入向量；dN _dZ 表示导数 d*N*/dZ。

神经网络工具箱中包含了一个自定义网络输入导数函数 mydnif()，输入 help mydnif 就可以获得有关此函数的帮助信息。举例说明如何应用 mydnif()计算网络输入导数。

【例 9-6】

```
Z1=rand(4,4);
Z2=rand(4,4);
Z3=rand(4,4);
N=mynif(Z1,Z2,Z3)
dN_dZ1=mydnif(Z1,N)
dN_dZ2=mydnif(Z2,N)
dN_dZ3=mydnif(Z3,N)
```

输出结果为

```
N =
    0.1414    0.1639    0.1650    0.2149
    0.1784    0.0493    0.0874    0.1371
    0.0449    0.2010    0.1945    0.1432
    0.2311    0.0395    0.0145    0.1821
dN_dZ1 =
    0.0016    0.0091    0.0051    0.0130
    0.0070    0.0004    0.0004    0.0077
    0.0000    0.0107    0.0127    0.0009
    0.0522    0.0002    0.0001    0.0048
dN_dZ2 =
    0.0123    0.0170    0.0047    0.0272
```

```
     0.0147    0.0000    0.0007    0.0177
     0.0004    0.0147    0.0289    0.0201
     0.0172    0.0000    0.0000    0.0206
dN_dZ3 =
     0.0038    0.0028    0.0151    0.0216
     0.0079    0.0022    0.0005    0.0008
     0.0001    0.0213    0.0073    0.0144
     0.0221    0.0003    0.0002    0.0131
```

输入 type mydnif，可以查看 mydnif() 的源程序。

```
>> type mydnif
function d = mydnif(z,n)
%MYDNIF Example custom net input derivative function of MYNIF.
%
%  Use this function as a template to write your own function.
%
%  Syntax
%
%    dN_dZ = dtansig(Z,N)
%      Z – S×Q matrix of Q weighted input (column) vectors.
%      N – S×Q matrix of Q net input (column) vectors.
%      dN_dZ – S×Q derivative dN/dZ.
%
%  Example
%
%    z1 = rand(4,5);
%    z2 = rand(4,5);
%    z3 = rand(4,5);
%    n = mynif(z1,z2,z3)
%    dn_dz1 = mydnif(z1,n)
%    dn_dz2 = mydnif(z2,n)
%    dn_dz3 = mydnif(z3,n)
% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.3.2.1 $
% ** Replace the following calculation with your
% **derivative calculation.
d = n.^2 .* z.^2;
% **Note that you have both the net input Z in question
% **and output N available to calculate the derivative.
```

可以将函数 mydnif() 作为一个模板，用来生成自定义的网络输入导数函数。

5. 权值函数

权值函数根据给定的输入向量（或者矩阵）***P*** 及权值矩阵 ***W***，计算一个加权的输入向量（或矩阵）***Z***。一旦定义了权值函数，就可以嵌入到网络中任意输入权值和层权值上。假设定义权值函数为 xiurw()，并嵌入网络第 1 个输入到第 3 层的权值上，则应用下面语句实现。

```
net.inputWeights{3,1}.weightFcn='xiurw'
```

这样，在网络进行仿真时，都可以应用此时设定的权值函数。

[Y, Pf, Af]=sim (net, P, Pi, Ai)

权值函数编制完成后，可以应用如下方式进行调用。

Z=xiurw (W, P)

其中，P 为输入向量；W 为权值矩阵；Z 为加权输入向量。

自定义权值函数返回如下信息。

- version：神经网络工具箱的版本。
- deriv：相关导数函数名。

神经网络工具箱中包含了一个自定义权值函数 mywf()，输入 help mywf 就可以获得有关此函数的帮助信息。举例说明如何应用 mywf()进行权值设置。

【例 9-7】

```
W=rand(1,4)
P=rand(4,1)
Z=mywf(W,P)
mywf('deriv')
```

输出结果为

```
W =
     0.1338     0.2071     0.6072     0.6299
P =
     0.3705
     0.5751
     0.4514
     0.0439
Z =
     0.2118
```

输入 type mywf，可以查看 mywf()的源程序。

```
>> type mywf
function z=mywf(w,p)
%MYWF Example custom weight function.
%
%   Use this function as a template to write your own function.
%
%   Calculation Syntax
%
%     Z = mywf(W,P)
%       W - S×R weight matrix.
%       P - R×Q matrix of Q input (column) vectors.
%       Z - S×Q matrix of Q weighted input (column) vectors.
%
%   Information Syntax
%
%     info = mytf(code) returns useful information for each CODE string:
```

```
%          'version' - Returns the Neural Network Toolbox version (3.0).
%          'deriv'     - Returns the name of the associated derivative function.
%
%   Example
%
%       w = rand(1,5);
%       p = rand(5,1);
%       z = mywf(w,p)
% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.2.2.1 $

if nargin < 1, error('Not enough arguments.'); end
if isstr(w)
   switch (w)
      case 'version'
      a = 3.0;            % <-- Must be 3.0.
      case 'deriv'
      a = 'mydtf';      % <-- Replace with the name of your
                              % associated function or ''
      otherwise
         error('Unrecognized code.')
   end
else
% **    Replace the following calculation with your
% **    weighting calculation.    The only constraint, if you
% **    want to define a derivative function, is that Z must
% **    be a sum of i terms, where each ith term is only a
% **    a function of w(i) and p(i).
   z = w* (p.^2);
end
```

可以将函数 mywf() 作为一个模板，用来生成自定义的权值函数。

6. 权值导数函数

如果自定义权值函数需要回传，则需要自定义权值导数函数，权值导数函数根据输入和权值计算此层的加权输入的导数。假设定义权值导数函数为 xiudrf()，则应用下面语句计算此层加权输入的导数。

dZ_dP=xiudrf ('p', W, P, Z)

dZ_dW=xiudrf ('w', W, P, Z)

其中，W 为权值矩阵；P 为输入向量；Z 为加权输入向量；dZ_dP 表示导数 **dZ/d*P***；dZ_dW 表示导数 **dZ/d*W***。

神经网络工具箱中包含了一个自定义权值导数函数 mydwf()，输入 help mydwf 就可以获得有关此函数的帮助信息。举例说明如何应用 mydwf() 计算加权输入导数。

【例 9-8】

```
W=rand(1,4)
```

```
P=rand(4,1)
Z=mywf(W,P)
dZ_dP=mydwf('p',W,P,Z)
dZ_dW=mydwf('w',W,P,Z)
```

输出结果为

```
W =
    0.7176    0.6927    0.0841    0.4544
P =
    0.4418
    0.3533
    0.1536
    0.6756
Z =
    0.4359
dZ_dP =
    0.6341    0.4894    0.0258    0.6140
dZ_dW =
    0.1952
    0.1248
    0.0236
    0.4565
```

输入 type mydwf，可以查看 mydwf()的源程序。

```
>> type mydwf
function d=mydwf(code,w,p,z)
%MYDWF Example custom weight derivative function for MYWF.
%
%  Use this function as a template to write your own function.
%
%  Syntax
%
%    dZ_dP = mydwf('p',W,P,Z)
%    dZ_dW = mydwf('w',W,P,Z)
%      W - S×R weight matrix.
%      P - R×Q matrix of Q input (column) vectors.
%      Z - S×Q matrix of Q weighted input (column) vectors.
%      dZ_dP - S×R derivative dZ/dP.
%      dZ_dW - R×Q derivative dZ/dW.
%
%  Example
%
%    w = rand(1,5);
%    p = rand(5,1);
%    z = mywf(w,p)
%    dz_dp = mydwf('p',w,p,z)
%    dz_dw = mydwf('w',w,p,z)
```

```
% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.2.2.1 $
% **   Replace the following calculations with your
% **   derivative calculation.   The only constraint is that
% **   the weight function must be a sum of elements, where
% **   each element i is a function of w(i) and p(i) only.

switch code
  case 'p', d = 2*w.*p';
  case 'w', d = p.^2;
  otherwise, error(['Unrecognized code.'])
end
% **   Note that you have both the transfer functions input N and
% **   output A available, which can often allow a more efficient
% **   calculation of the derivative than with just N.
```

可以将函数 mydwf()作为一个模板，用来生成自定义的权值导数函数。

9.2.4　自组织函数

本小节介绍如何自建自组织特征映射网络中所用到的拓扑函数和距离函数。

1．拓扑函数

在指定维数的情况下，拓扑函数可以计算某一层中神经元的位置。一旦定义了拓扑函数，就可以嵌入到网络任意一层上。假设定义了拓扑函数为 xiutop ()，并嵌入网络第 3 层，则应用下面语句实现。

net.layers{2}.topologyFcn='xiutop';

这样，任何时候网络进行训练或者自适应调节时，都可以应用此时设定的拓扑函数。

[net, tr]=train (NET, P, T, Pi, Ai)

[net, Y, E, Pf, Af]=adapt (NET, P, T, Pi, Ai)

拓扑函数编制完成后，输入一定的维数。调用拓扑函数，就可以计算出神经元的位置 pos，调用方式如下。

pos=xiutop (dim1, dim2, ... , dimN)

神经网络工具箱中包含一个自定义拓扑函数 mytopf()，输入 help mytopf 就可以获得有关此函数的帮助信息。举例说明如何应用 mytopf()拓扑函数计算神经元的位置。

【例 9-9】绘制应用 mytopf()函数计算神经元位置的拓扑结构图，如图 9-5 所示。

```
pos=mytopf(15,15);
plotsom(pos)
```

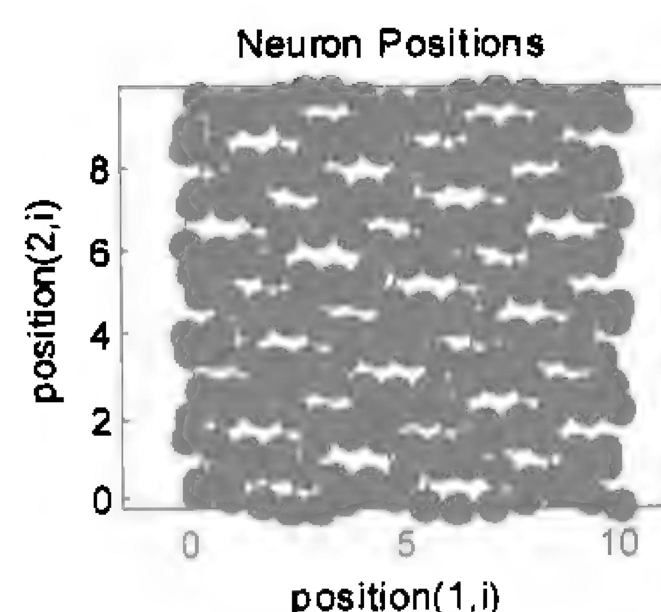


图 9-5　拓扑结构

输入 type mytopf，可以查看 mytopf()的源程序。

```
>> type mytopf
function pos=mytopf(varargin)
%MYTOPF Example custom topology function.
%
%   Use this function as a template to write your own function.
%
%   Syntax
%
%     pos = mytopf(dim1,dim2,...,dimN)
%       dimi - number of neurons along the ith layer dimension
%       pos  - N×S matrix of S position vectors, where S is the
%              total number of neurons which is defined by the
%               product dim1*dim1*…*dimN.
%
%   Example
%
%     pos = mytopf(20,20);
%     plotsom(pos)

% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.2.2.1 $

% ** Replace the code below with your own calculation
% ** for the neuron positions.

dim = [varargin{:}];       % The dimensions as a row vector
size = prod(dim);          % Total number of neurons
dims = length(dim);        % Number of dimensions
pos = zeros(dims,size);    % The size that POS will need to be
len = 1;
pos(1,1) = 0;
for i=1:length(dim)
  dimi = dim(i);
  newlen = len*dimi;
  pos(1:(i-1),1:newlen) = pos(1:(i-1),rem(0:(newlen-1),len)+1);
  posi = 0:(dimi-1);
  pos(i,1:newlen) = posi(floor((0:(newlen-1))/len)+1);
  len = newlen;
end
for i=1:2
  pos(i,:)=pos(i,:)*0.7+sin([1:size]*exp(1)/5*i)*0.2;
end
```

可以将函数 mytopf()作为一个模板，用来生成自定义的拓扑函数。

2. 距离函数

在指定位置的情况下，距离函数可以计算某一层中神经元的距离。一旦定义了距离函数，就可以嵌入到网络任意一层上。假设定义了距离函数为 xiudistf()，并嵌入网络第 3 层，则应用下面语句实现。

net.layers{3}.distanceFcn='xiudistf ';

这样，任何时刻网络进行训练或者自适应调节时，都可以应用此时设定的距离函数。

[net, tr]=train (NET, P, T, Pi, Ai)

[net, Y, E, Pf, Af]=adapt (NET, P, T, Pi, Ai)

距离函数编制完成后，输入一定的位置信息。调用距离函数，就可以计算出距离 d，调用方式如下。

d=xiudistf (pos1, pos2, ... , posN)

神经网络工具箱中包含一个自定义距离函数 mydistf()，输入 help mydistf 就可以获得有关此函数的帮助信息。举例说明如何应用 mydistf()距离函数计算神经元之间的距离。

【例 9-10】

```
pos=gridtop(2,3);
d=mydistf(pos)
```

输出结果为

```
d =
         0    1.0000    1.0000    1.5874    2.0000    2.4473
    1.0000         0    1.5874    1.0000    2.4473    2.0000
    1.0000    1.5874         0    1.0000    1.0000    1.5874
    1.5874    1.0000    1.0000         0    1.5874    1.0000
    2.0000    2.4473    1.0000    1.5874         0    1.0000
    2.4473    2.0000    1.5874    1.0000    1.0000         0
```

输入 type mydistf，可以查看 mydistf()的源程序。

```
>> type mydistf
function d = mydistf(pos)
%MYDISTF Example custom distance function.
%
%   Use this function as a template to write your own function.
%
%      Syntax
%
%     d = mydistf(pos)
%        pos - N×S matrix of S neuron position vectors.
%        d     - S×S matrix of neuron distances.
%
%   Example
%
%     pos = gridtop(3,2);
%     d = mydistf(pos)
% Copyright 1997 The MathWorks, Inc.
```

```
% $Revision: 1.3.2.1 $

s = size(pos,2);
for i=1:s
  for j=1:s
% ** Replace the following line of code with your own
% ** measure of distance.
    d(i,j) = norm(pos(:,i)-pos(:,j),1.5);
  end
end
```

可以将函数 mydistf() 作为一个模板，用来生成自定义的距离函数。

第 10 章　神经网络的应用

10.1　线性神经网络在线性预测中的应用

【例 10-1】利用 adapt 函数对自适应滤波网络进行训练，并用设计好的自适应滤波网络对某时变正弦信号进行自适应预测。

1）定义输入矢量和目标矢量

待预测的时变正弦信号 T 定义如下

$$T=\begin{cases}\sin(4\pi k) & k\leqslant 4\text{s}\\ \sin(8\pi k) & 4\text{s}<k\leqslant 6\text{s}\end{cases}$$

可见，4s 以后正弦信号频率加倍。此外，信号的采样频率前 4s 取每秒采样 20 次，4s 以后加倍。该时变正弦信号 T 可以采用如下 MATLAB 语句生成。

```
t1=0:0.05:4;
t2=4.04:0.024:6;
t=[t1 t2];
T=[sin(t1*4*pi) sin(t2*8*pi)];
```

为了便于训练，将信号 T 转换成序列的形式。

```
T=con2seq(T);
```

令网络输入信号与目标序列相同。

```
P=T;
```

2）网络设计

在每一个采样时刻点，采用该时刻之前的 5 个采样信号值作为网络的输入，网络的输出则为下一时刻信号的预测值。据此，线性自适应滤波网络可以按如图 10-1 所示的结构进行设计。

首先，采用 newlin 函数生成如图 10-1 所示的自适应滤波网络，即

```
lr=0.1;
delays=[1 2 3 4 5];
net=newlin(minmax(cat(2,P{:})),1,delays,lr);
```

其中，网络学习速率 lr 取为 0.1。

然后，利用 adapt 函数对所生成的神经网络进行训练。

```
[net,Y,E]=adapt(net,P,T);
```

训练完成之后，即返回设计好的自适应滤波网络。

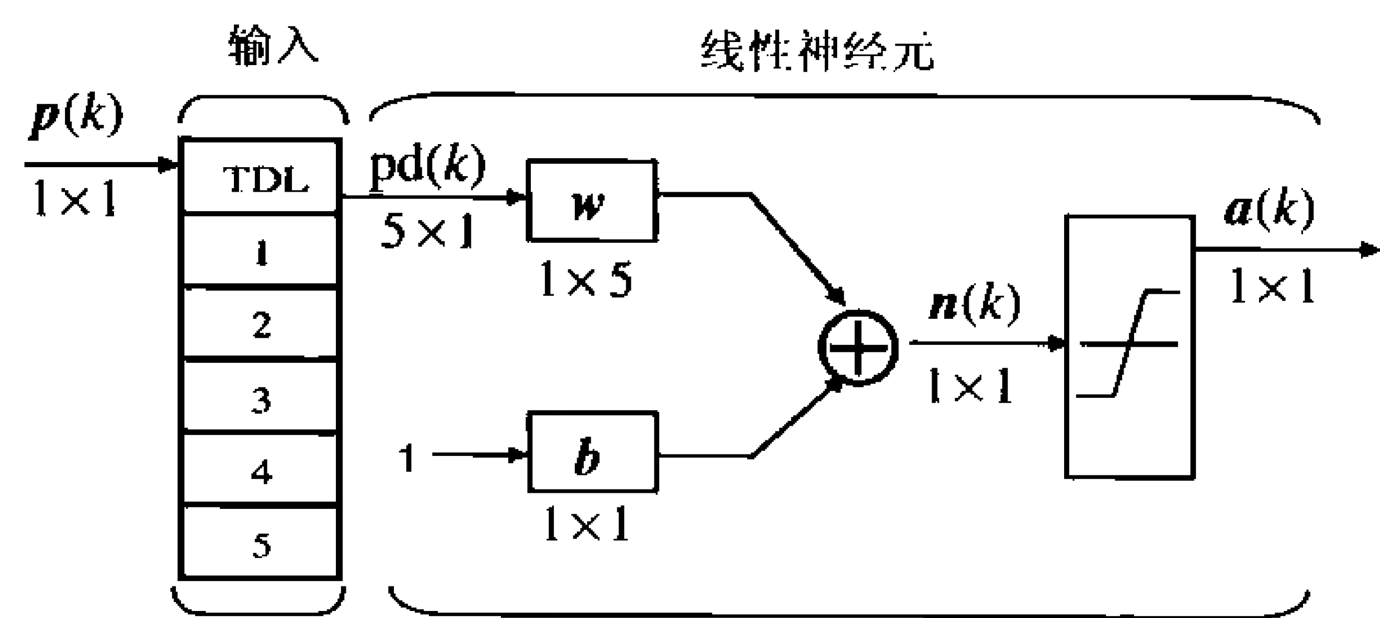


图 10-1　自适应滤波网络结构图

3）网络测试

为了检测网络的预测效果，在网络训练完成之后，可以将网络预测输出与目标信号绘制在同一幅图中加以比较，如图 10-2 所示，其中虚线代表目标信号。

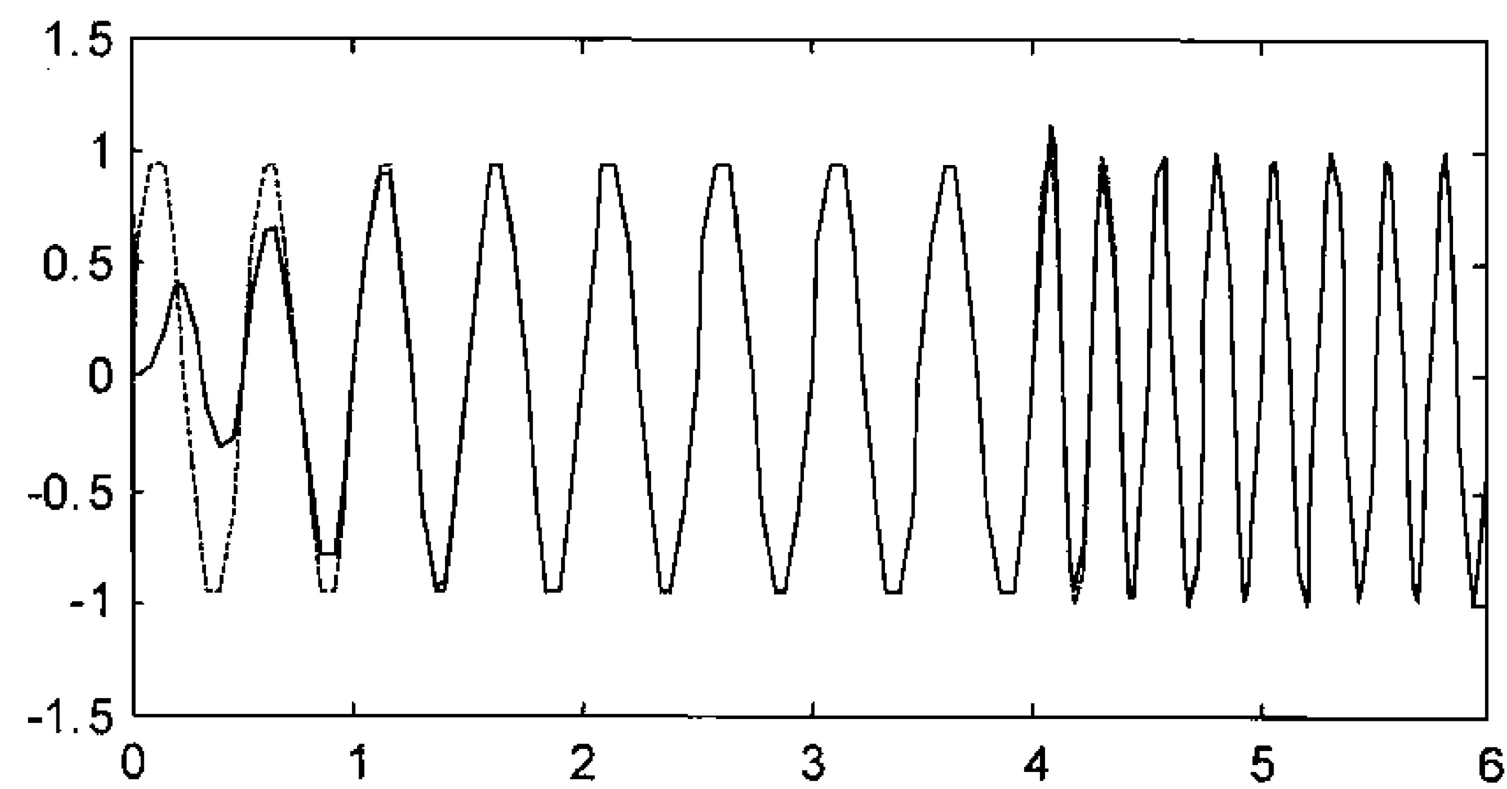


图 10-2　预测信号与目标信号曲线

由图 10-2 可见，在经过大约 1.5s 的自适应训练之后，网络几乎可以完全跟踪目标信号（曲线几乎完全重合）。在第 4s 时刻，由于目标信号的频率发生突变，所以网络的输出与目标信号曲线稍有偏差，但由于神经网络先前已经对与当前信号相似的正弦信号进行了学习，所以经过更短的训练时间就能再次精确地预测目标信号。

图 10-3 给出了网络的预测误差曲线。

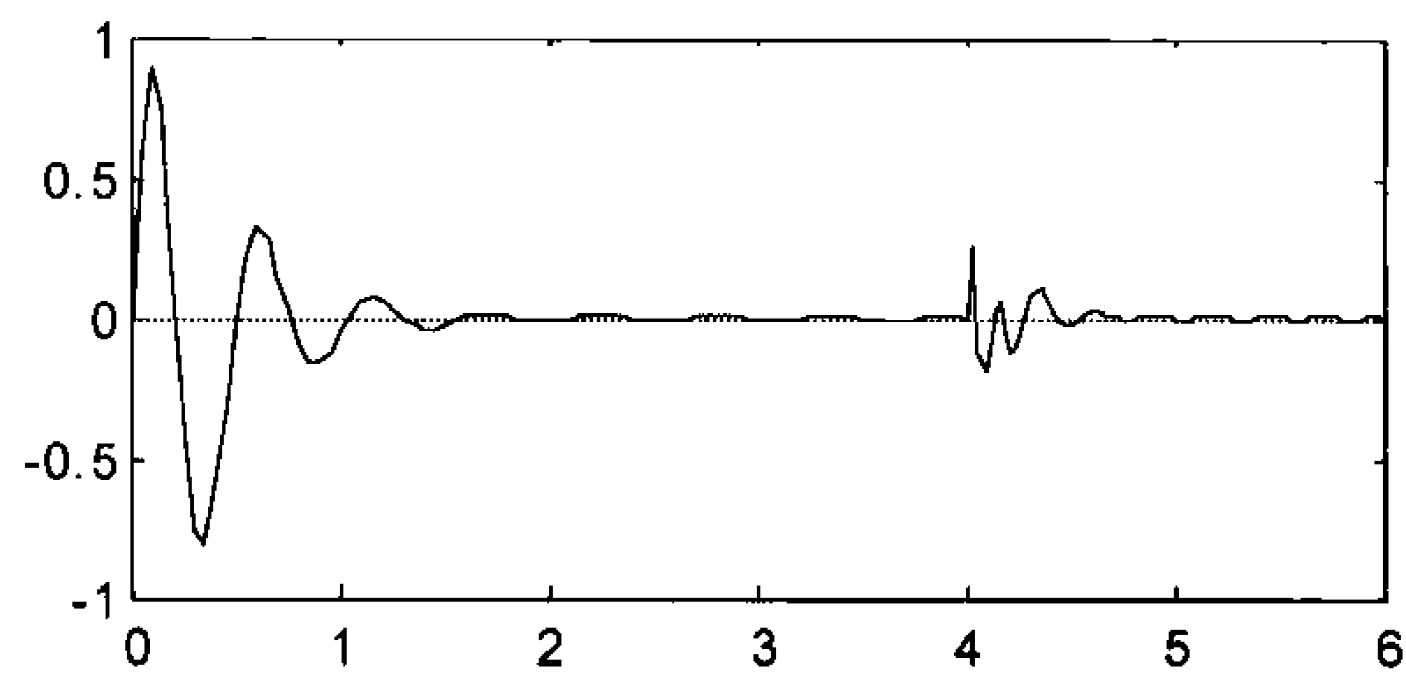


图 10-3　预测误差曲线

4）本例完整的 MATLAB 程序代码

```
clear all
%定义输入矢量和目标矢量
t1=0:0.05:4;
t2=4.04:0.024:6;
t=[t1 t2];
T=[sin(t1*4*pi) sin(t2*8*pi)];
T=con2seq(T);
%目标信号
```

```
P=T;
%绘制待预测的目标信号曲线
plot(t,cat(2,T{:}))
%生成自适应滤波网络
%学习速率
lr=0.1;
delays=[1 2 3 4 5];
net=newlin(minmax(cat(2,P{:})),1,delays,lr);
%对网络进行训练
[net,Y,E]=adapt(net,P,T);
%绘制网络预测输出曲线
plot(t,cat(2,T{:}),'--',t,cat(2,Y{:}),'r')
clf
%绘制预测误差曲线
plot(t,cat(2,E{:}),[min(t) max(t)],[0 0],':r')
```

10.2　神经模糊控制在洗衣机中的应用

20 世纪 90 年代初期，日本松下公司推出了神经模糊控制全自动洗衣机。这种洗衣机能够自动判断衣物的质地软硬程度、衣量多少、脏污程度和性质等，应用神经模糊控制技术，自动生成模糊控制规则和隶属度函数，预设洗衣水位、水流强度和洗涤时间，在整个洗衣过程中实时调整这些参数，达到最佳的洗衣效果。

10.2.1　洗衣机的模糊控制

洗衣机的主要被控参量为洗涤时间和水流强度，而影响这一输出参量的主要因子是被洗物的浑浊程度和浑浊性质，后者可用浑浊度的变化率来描述。在洗涤过程中，油污的浑浊度变化率小，泥污的浑浊度变化率大。因此，浑浊度及其变化率可以作为控制系统的输入参量，而洗涤时间和水流强度可作为控制量，即系统的输出。实际上，洗衣过程中的这类输入和输出之间很难用一定的数学模型进行描述。系统运行过程中具有较大的不确定性，控制过程在很大程度上依赖操作者的经验，这样一来，利用常规的方法进行控制难以奏效。但是，如果利用专家知识进行控制决策，往往容易实现优化控制，这就是在洗衣机中引入模糊控制技术的主要原因之一。

根据上述的模糊控制基本原理，可得出确定洗涤时间的模糊推理框图，如图 10-4 所示。其输入量为水的浑浊度及其变化率，输出量为洗涤时间。考虑到适当的控制性能需要和程序简化，定义输入量浑浊度的模糊词集为{清、较浊、浊、很浊}，定义浑浊度变换率的模糊词集为{零、小、中、大}，定义输出变量洗涤时间的模糊词集为{短、较短、标准、长}。描述输入/输出变量的词集都具有模糊特性，可以用模糊集合表示。因此，模糊概念的确定问题就直接转换为求取模糊集合的隶属函数问题。

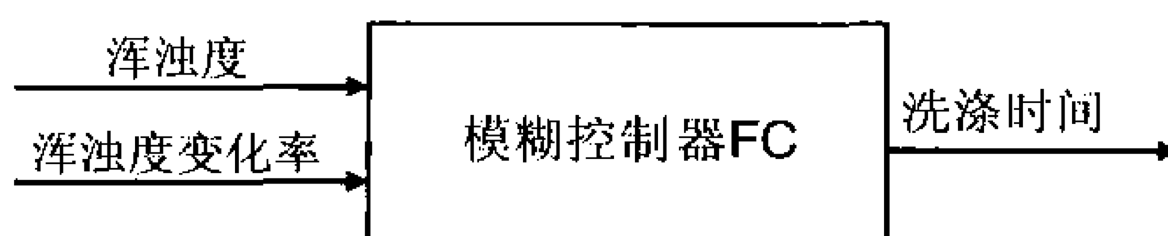


图 10-4　确定洗涤时间的模糊推理框图

通常定义一个模糊子集，实际上就是确定模糊子集隶属函数形状的过程。将确定的隶属函数曲线离散化，就得到了有限个点上的隶属度，构成了一个相应的模糊子集。如图 10-5 所示，定义了隶属函数曲线表示论域 $\boldsymbol{x}$ 中元素对模糊变量 A 的隶属程度。设定该隶属函数的论域 $\boldsymbol{x}$ 为

$$\boldsymbol{x}=(-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6)$$

则有

$$\mu_A(2)=\mu_A(6)=0.2\ ,\quad \mu_A(3)=\mu_A(5)=0.7\ ,\quad \mu_A(4)=1$$

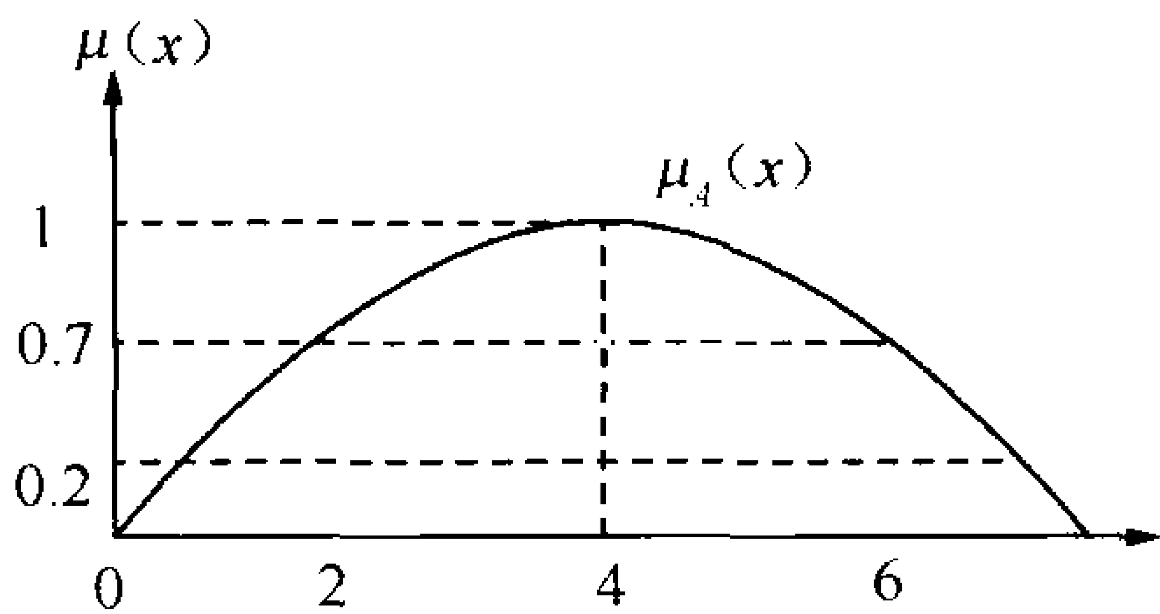
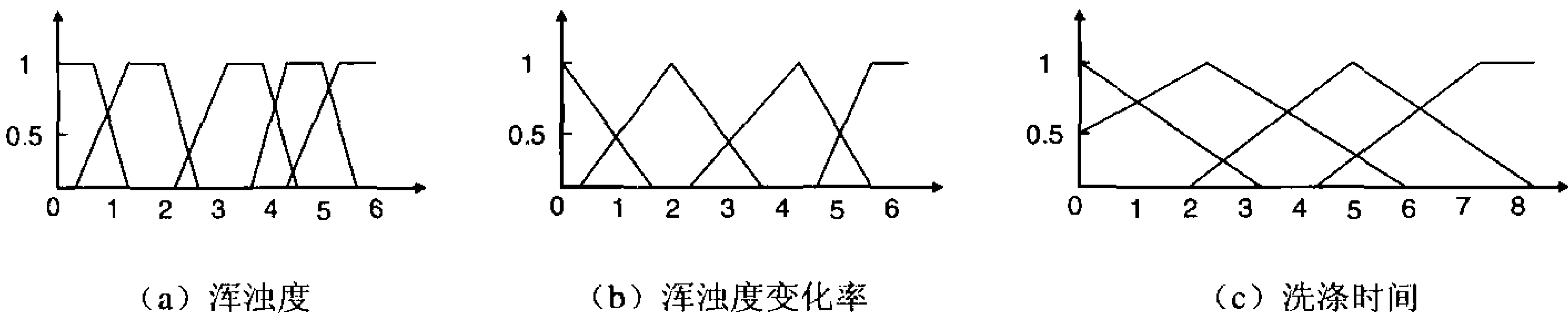


图 10-5　模糊变量 A 的隶属度函数

论域 x 中除了 2、3、4、5、6 外，各点的隶属度均为 0。那么模糊变量 A 的模糊子集为 A=0.2/2+0.7/3+1/4+0.7/5+0.2/6。

通过这个例子可以看出，在隶属函数的曲线确定后，就可以很容易地定义出一个模糊变量的模糊子集。洗衣机模糊控制的输入和输出变量的隶属函数如图 10-6 所示，由此可以相继确定它们的模糊子集。



（a）浑浊度　　（b）浑浊度变化率　　（c）洗涤时间

图 10-6　洗衣机模糊变量隶属函数

洗衣机的模糊控制规则可以归纳为 16 条，如表 10-1 所示。

表 10-1　洗衣机的模糊控制规则表

洗涤时间		浑浊度			
		清	较浊	浊	很浊
变化率	零	短	较短	标准	标准
	小	标准	标准	标准	标准
	中	标准	长	长	长
	大	标准	标准	长	长

10.2.2　洗衣机的神经网络模糊控制器的设计

洗衣机模糊控制的控制部分框图如图 10-7 所示，模糊控制器如图中虚线所示。模糊控制的过程是这样的：首先洗衣机获取的浑浊度信息由传感器送到信息处理单元，分为浑浊度和浑浊度变化率送入模糊控制器。对于输入的模糊量，需要将其转换成模糊变量，通过单片机，利用

查表法按照模糊推理法则做出决策，结果被认为是模糊变量，经过去模糊化单元处理，再由执行机构去修改洗涤时间，这样就完成了一次模糊控制算法过程。

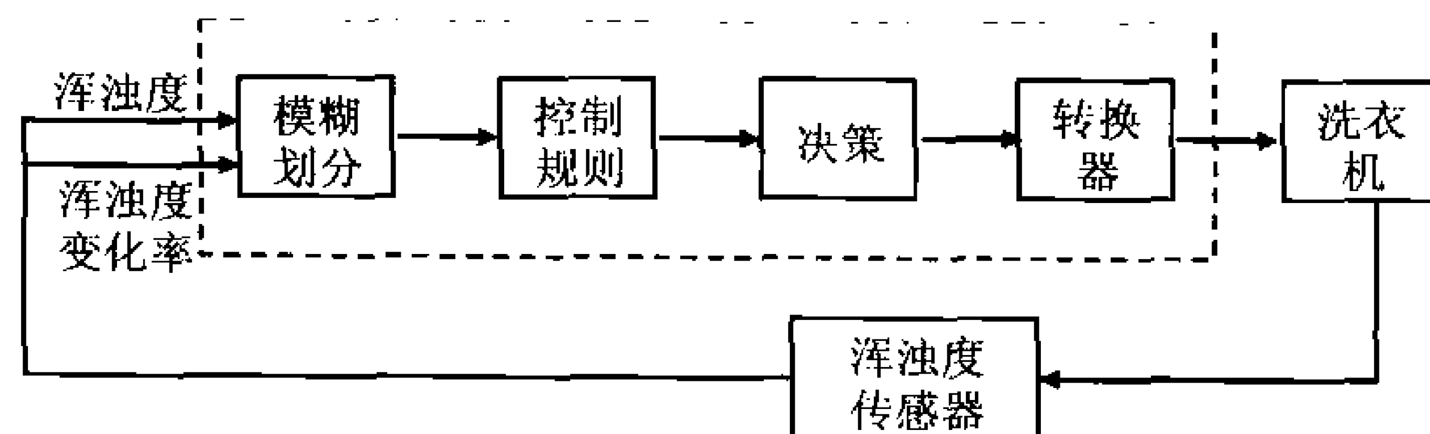


图 10-7 模糊控制的控制部分框图

一般的模糊控制洗衣机将“专家经验”通过模糊控制规则表体现出来，运行中通过查表做出控制决策，这比需要操作者设定程序的电脑控制洗衣机前进了一大步。但是，这种洗衣机由于规则表需要占用大量的内存空间，查表反应速度慢，只能够按照已经编入的规则进行控制，因此不够理想。而把神经网络和模糊控制相结合，则能够解决这些问题。

洗衣机的神经网络模糊控制是利用离线训练好的网络，通过在线计算即可得到最佳输出。这种控制模式的反应速度快，而且神经网络又具有自学习功能和联想能力，对于未在训练中出现的样本，也可以通过联系记忆的功能，做出控制决策，表现非常灵活。

洗衣机的神经网络模糊控制器的控制系统中含有多个神经模糊环节，下面仅介绍以浑浊度和浑浊度变化率为输入变量来确定洗涤时间的控制器。控制器的控制框图如图 10-8 所示，浑浊度神经网络结构如图 10-9 所示。

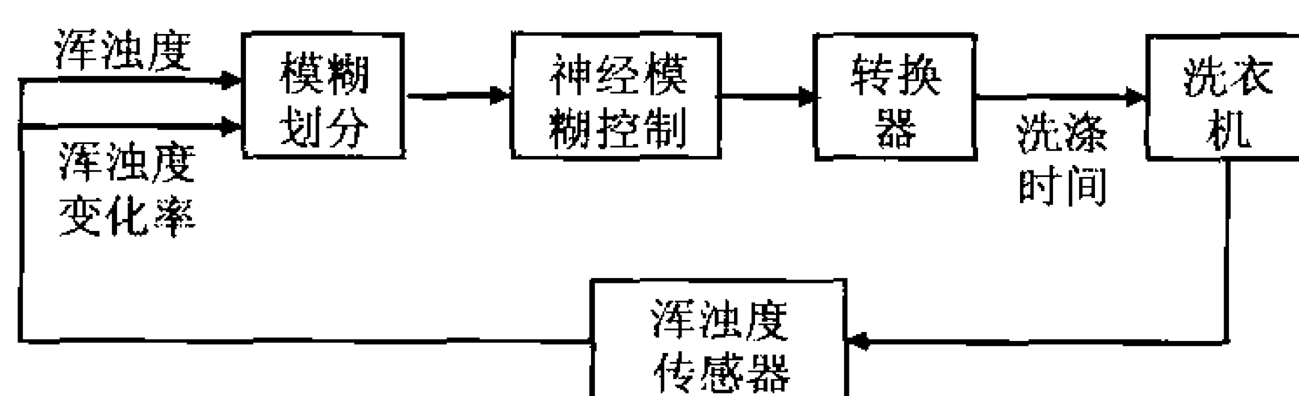


图 10-8 神经网络模糊控制器的控制框图

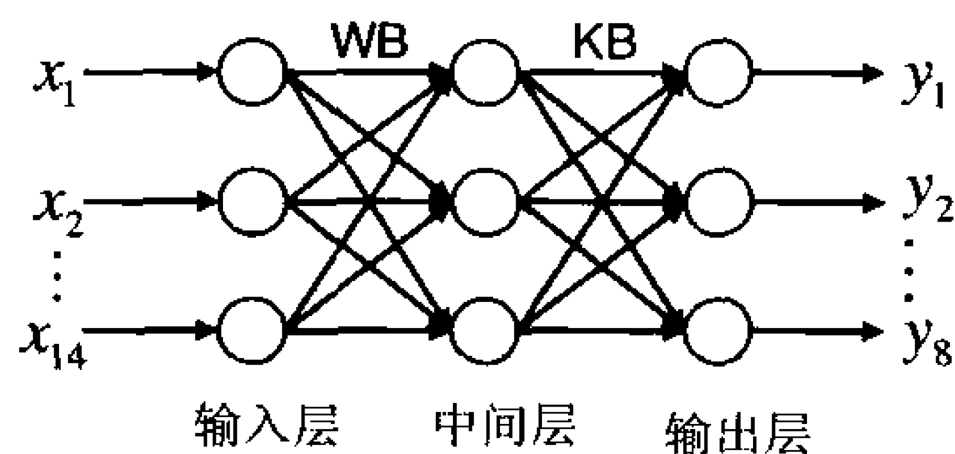


图 10-9 浑浊度神经网络结构

神经模糊控制器在输入/输出参量的选择及模糊论域和模糊子集的确定方面，与一般模糊控制器没有什么区别，只是在推理手段上引入了神经网络。令 $x_1 \sim x_7$ 为输入量浑浊度的模糊子集，$x_8 \sim x_{14}$ 为输入量浑浊度变化率的模糊子集，$y_1 \sim y_8$ 为输出控制量的模糊子集。从模糊控制规则表 10-1 中可以看出，其有 16 条控制规则，每条规则都是一对样本，则共有 16 对样本。例如，当浑浊度为“清”，浑浊度变化率为“零”时，洗涤时间应该为“短”，这个样本可以表示为

$$x=[1,0.6,0.1,0,0,0,0,1.0,0.5,0,0,0,0,0]^{\mathrm{T}}, \quad y=[1.0,0.5,0,0,0,0,0,0]^{\mathrm{T}}$$

其中，x 中的各元素为对应的隶属函数，即模糊子集的赋值。同理可列出其他 15 个样本对，并将它们依次送入神经网络进行离线训练，当训练结束后，神经网络已经记忆了模糊控制规则，使用时具有联想记忆功能。每一个输入参量的模糊量如表 10-2 所示。

表 10-2 输入参量的模糊量

输入参量		模 糊 量
浑浊度	清	1 0.6 0.1 0 0 0 0
	较浊	0 0.6 0.6 0 0 0 0
	浊	0 0 0.6 1 0 0 0
	很浊	0 0 0 0 1 0.6 0
浑浊度变化率	零	1 0.5 0 0 0 0 0
	小	0 0.5 1 0.4 0 0 0
	中	0 0 0 0.4 1 0.6 0
	大	0 0 0 0 0 0 0.8
洗涤时间	短	1 0.5 0 0 0 0 0 0
	较短	0.4 0.8 1 0.8 0.4 0.2 0 0
	标准	0 0 0 0.2 0.6 1 0.6 0.2
	长	0 0 0 0 0 0.2 0.5 0.8

根据模糊规则，可以得到网络的训练样本 ***P*** 和 ***T***。完整的 MATLAB 代码如下。

```
P=[1 0.6 0.1 0 0 0 0 1 0.5 0 0 0 0 0;
   1 0.6 0.1 0 0 0 0 0 0.5 1 0.4 0 0 0;
   1 0.6 0.1 0 0 0 0 0 0 0 0.4 1 0.6 0;
   1 0.6 0.1 0 0 0 0 0 0 0 0 0 0 0.8;
   0 0.6 0.6 0 0 0 0 1 0.5 0 0 0 0 0;
   0 0.6 0.6 0 0 0 0 0 0.5 1 0.4 0 0 0;
   0 0.6 0.6 0 0 0 0 0 0 0 0.4 1 0.6 0;
   0 0.6 0.6 0 0 0 0 0 0 0 0 0 0 0.8;
   0 0 0.6 1 0 0 0 1 0.5 0 0 0 0 0;
   0 0 0.6 1 0 0 0 0 0.5 1 0.4 0 0 0;
   0 0 0.6 1 0 0 0 0 0 0 0.4 1 0.6 0;
   0 0 0.6 1 0 0 0 0 0 0 0 0 0 0.8;
   0 0 0 0 1 0.6 0 1 0.5 0 0 0 0 0;
   0 0 0 0 1 0.6 0 0 0.5 1 0.4 0 0 0;
   0 0 0 0 1 0.6 0 0 0 0 0.4 1 0.6 0;
   0 0 0 0 1 0.6 0 0 0 0 0 0 0 0.8]';
T=[1 0.5 0 0 0 0 0 0;
   0 0 0 0.2 0.6 1 0.6 0.2;
   0 0 0 0.2 0.6 1 0.6 0.2;
   0 0 0 0.2 0.6 1 0.6 0.2;
   0.4 0.8 1 0.8 0.4 0.2 0 0;
   0 0 0 0.2 0.6 1 0.6 0.2;
   0 0 0 0 0 0.2 0.5 0.8;
   0 0 0 0.2 0.6 1 0.6 0.2;
   0 0 0 0.2 0.6 1 0.6 0.2;
   0 0 0 0.2 0.6 1 0.6 0.2;
   0 0 0 0 0 0.2 0.5 0.8;
```

```
    0 0 0 0 0 0.2 0.5 0.8;
    0 0 0 0.2 0.6 1 0.6 0.2;
    0 0 0 0.2 0.6 1 0.6 0.2;
    0 0 0 0 0 0.2 0.5 0.8;
    0 0 0 0 0 0.2 0.5 0.8]';
%根据 Kolmogorov 定理，由于输入层有 14 个节点，所以中间层有 29 个节点
%中间层神经元的传递函数为 tansig
%输出层有 8 个节点，其神经元传递函数为 logsig
%训练函数采用 traingdx
net=newff(minmax(P),[29,8],{'tansig','logsig'},'traingdx');
%训练步数为 1000 次
%训练目标误差为 0.001
net.trainParam.epochs=1000;
net.trainParam.goal=0.001;
net=train(net,P,T);
Y=sim(net,P);
%求训练值在每一个点上的误差
for i=1:16
    x(i)=norm(Y(:,i));
end
plot(1:16,x);
```

网络的误差曲线如图 10-10 所示。由此可见，网络的最大误差不超过 0.2，说明网络性能是可以满足控制要求的。

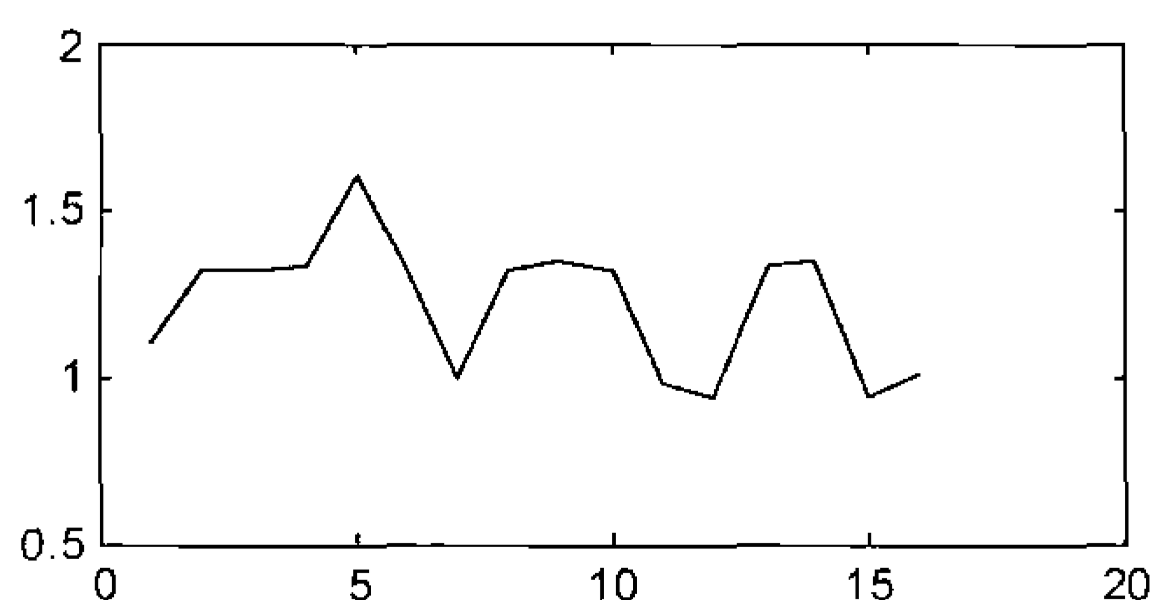


图 10-10　网络误差曲线

网络训练结果为

```
TRAINGDX, Epoch 0/1000, MSE 0.28117/0.001, Gradient 0.082897/1e-006
TRAINGDX, Epoch 25/1000, MSE 0.278678/0.001, Gradient 0.0830836/1e-006
TRAINGDX, Epoch 50/1000, MSE 0.270858/0.001, Gradient 0.082787/1e-006
TRAINGDX, Epoch 75/1000, MSE 0.246485/0.001, Gradient 0.0731321/1e-006
TRAINGDX, Epoch 100/1000, MSE 0.201249/0.001, Gradient 0.0454135/1e-006
TRAINGDX, Epoch 125/1000, MSE 0.127164/0.001, Gradient 0.0455586/1e-006
TRAINGDX, Epoch 150/1000, MSE 0.0361097/0.001, Gradient 0.0142122/1e-006
TRAINGDX, Epoch 175/1000, MSE 0.0125535/0.001, Gradient 0.00572951/1e-006
TRAINGDX, Epoch 200/1000, MSE 0.00937949/0.001, Gradient 0.0020679/1e-006
TRAINGDX, Epoch 225/1000, MSE 0.00223234/0.001, Gradient 0.00983431/1e-006
TRAINGDX, Epoch 244/1000, MSE 0.000978229/0.001, Gradient 0.00205187/1e-006
TRAINGDX, Performance goal met.
```

网络经过 244 次训练后，目标误差达到要求，结果如图 10-11 所示。

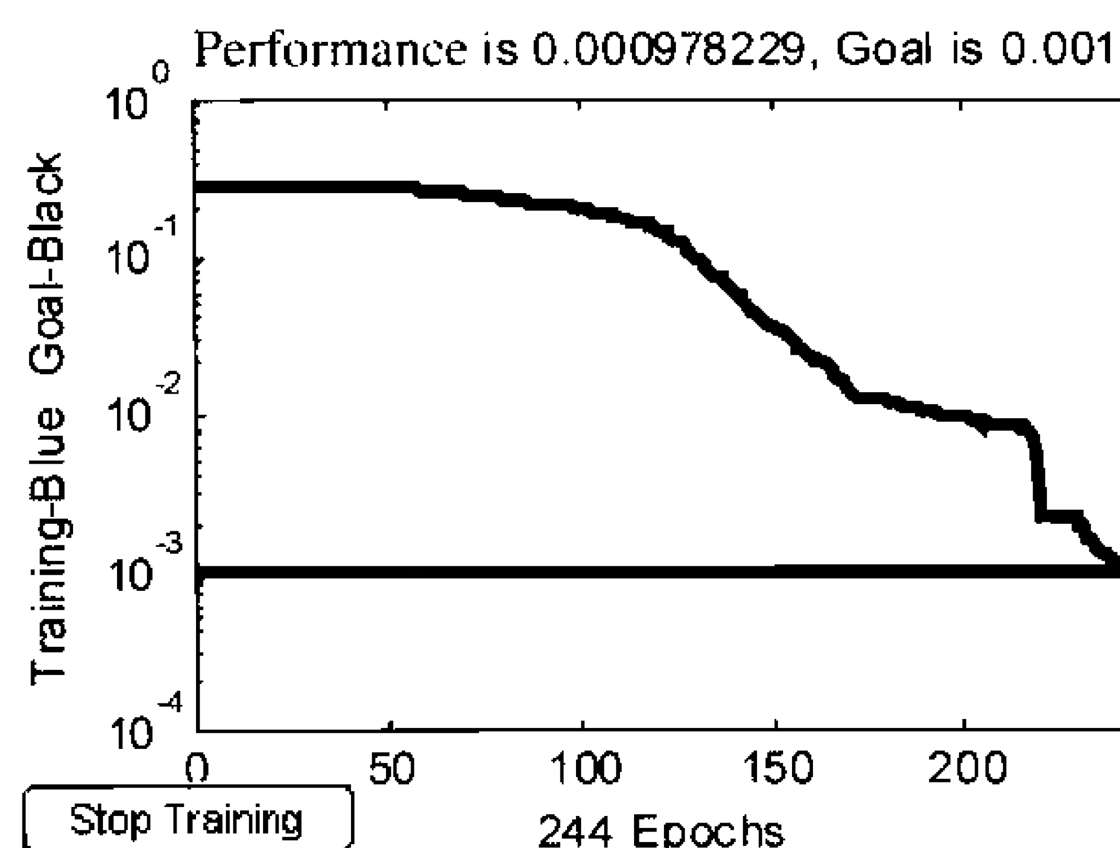


图 10-11　训练结果

10.3　模糊神经网络在配送中心选址中的应用

合理的物流配送中心选址能节省费用，加快货物的流通，增加物流企业的收益，因此，物流配送中心的选址决策对于整个物流系统的优化是个十分重要的问题。

10.3.1　问题概述

长期以来，对于物流配送中心选址，人们提供了一种新的决策方法，如模糊综合评判法、AHP 层次分析法及结合层次法的模糊排序方法等。但这些方法也有一些缺点，利用模糊综合评判法，其指标权重难以确定；用专家打分法确定权重，人为因素又过重；利用层次分析法确定权重可以弱化人为因素，但是层次分析法要求指标的层次结构系统中的要素互相独立，否则就不能应用这种方法，而这些指标之间往往存在依赖关系，如地价和运输条件、政府政策和经营环境等。

模糊综合评价作为一种多属性的综合评价方法，其隶属函数权重有一定主观性，因此它的应用就有局限性。而 BP 算法则可以较客观地评价不同的方案，将模糊方法应用到 BP 算法的输入值中。综合两种算法的优势，通过网络学习得到选址的优化度，进而得到对于选址方案的一个较为准确的评价。

10.3.2　设计

物流配送中心的选址通常是在一定的原则如降低成本原则、经济效益原则、提高客户服务水平原则等的指导之下，预先选择一些方案，然后再通过各种方法对这些方案进行比较，最终从中选出满意的一个或几个方案作为新的中心地址。影响配送中心选址的因素很多，可以根据物流学的原理，结合自身的实际情况，选择其中较重要的一些因素作为决策指标。这些因素大致可分为自然环境、交通运输条件、经营环境、候选地条件及公共设施几大类。指标选择的好坏对正确决策相当重要。选址主要考虑几个因素：客户的分布、供应商的分布、交通条件、用地条件、自然地理条件和环境条件等。

采用模糊方法计算影响因素的隶属度作为神经网络的输入，在 n 个影响因素中，既有定量因素也有定性因素，对不同的因素要用不同的方法来确定隶属度。对定量因素可以采用常见模糊分布来确定隶属函数，例如，模糊约束集合运输距离最小符合的模糊分布-降半梯形分布。对于定性因素而言，首先要用专家咨询法、专家评分法等方法进行量化，再利用相关的隶属函数确定隶属度。这里考虑的定量因素有：候选地地价、运输距离和候选地面积。其中运输距离

的隶属度通过上文介绍的方法确定，而另两种定量指标中，候选地地价与运输距离的隶属函数相同，模糊约束集合候选地面积适中符合模糊分布-正态分布，对应的隶属函数为

$$u(x) = \mathrm{e}^{-k(x-c)^2} \quad (k, c > 0)$$

而定性因素有：客户分布、供应商分布、地质条件、通信条件、道路设施。定性因素首先对不同的情况做出分类描述，再将自然语言的评价分类转换为相应的隶属度，如表 10-3 所示。

表 10-3　定性指标的隶属函数

指　标	分类描述	模糊约束集	隶 属 度
客户分布	集中、一般、分散	客户分布集中 A1	A1={1, 0.5, 0}
供应商分布	集中、一般、分散	供应商分布集中 A2	A2={1, 0.5, 0}
通信条件	很好、好、一般、差、很差	通信条件良好 A3	A3={1, 0.8, 0.5, 0.2, 0}
地质条件	很好、好、一般、差、很差	地质条件良好 A4	A4={1, 0.8, 0.5, 0.2 0}

将指标隶属度输入到网络中，将专家值作为网络的期望输出。表 10-4 所示为经过量化和模糊处理并求出隶属度的实验训练数据。表 10-5 所示为经过初步筛选后的实验数据及其方案的评价结果和排序。

表 10-4　实验训练数据

方案序号	地质条件	客户分布	候选地地价	供应商分布	运输距离	通信条件	候选地面积	道路设施	专家评价结果
1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	0.80	0.87	0.89	0.82	0.78	0.80	0.75	0.33	0.79
3	0.67	0.93	0.22	0.75	1.00	0.80	0.49	0.66	0.74
4	0.92	0.80	0.89	0.92	0.89	0.80	1.00	1.00	0.81
5	0.87	0.93	1.00	1.00	1.00	1.00	1.00	1.00	0.96
6	0.80	0.72	0.89	0.82	0.89	0.80	0.75	1.00	0.83
7	0.67	0.72	0.67	0.66	0.67	0.60	0.49	0.66	0.69
8	0.72	0.80	0.78	0.75	0.78	0.80	0.75	0.66	0.75
9	0.60	0.60	0.56	0.58	0.56	0.60	0.49	0.66	0.58
10	0.47	0.47	0.44	0.41	0.44	0.40	0.49	0.35	0.51

表 10-5　经筛选后的实验数据及方案的评价结果和排序

方案序号	地质条件	客户分布	候选地地价	供应商分布	运输距离	通信条件	候选地面积	道路设施	评价结果	排序
11	0.40	0.40	0.33	0.33	0.33	0.40	0.49	0.35	0.48	4
12	0.08	0.93	0.56	0.92	0.89	0.60	0.24	0.33	0.81	2
13	0.67	0.60	0.89	0.82	1.00	0.80	0.75	0.29	0.97	1
14	0.32	0.40	0.67	0.33	0.33	0.80	0.75	0.35	0.54	3
15	0.87	0.72	0.89	0.92	0.89	0.40	0.49	0.29	0.47	5

根据模糊规则，可得到网络的训练样本 ***P*** 和 ***T***。完整的 MATLAB 代码如下。

```
P=[1.00    1.00 1.00 1.00 1.00 1.00 1.00 1.00;
   0.80    0.87 0.89 0.82 0.78 0.80 0.75 0.33;
   0.67    0.93 0.22 0.75 1.00 0.80 0.49 0.66;
   0.92    0.80 0.89 0.92 0.89 0.80 1.00 1.00;
   0.87    0.93 1.00 1.00 1.00 1.00 1.00 1.00;
   0.80    0.72 0.89 0.82 0.89 0.80 0.75 1.00;
   0.67    0.72 0.67 0.66 0.67 0.60 0.49 0.66;
   0.72    0.80 0.78 0.75 0.78 0.80 0.75 0.66;
   0.60    0.60 0.56 0.58 0.56 0.60 0.49 0.66;
   0.47    0.47 0.44 0.41 0.44 0.40 0.49 0.35]';
T=[1.00 0.79 0.74 0.81 0.96 0.83 0.69 0.75 0.58 0.51];
%根据 Kolmogorov 定理，由于输入层有 8 个节点，所以中间层有 17 个节点
%中间层神经元的传递函数为 tansig
%输出层有 1 个节点，其神经元传递函数为 logsig
%训练函数采用 trainlm
net=newff(minmax(P),[17,1],{'tansig','logsig'},'trainlm');
%训练步数为 1000 次
%训练目标误差为 0.001
net.trainParam.epochs=1000;
net.trainParam.goal=0.001;
net=train(net,P,T);
Y=sim(net,P);
```

网络的训练结果为

```
TRAINLM, Epoch 0/1000, MSE 0.05559/0.001, Gradient 1.4427/1e-010
TRAINLM, Epoch 3/1000, MSE 0.000339403/0.001, Gradient 0.0831561/1e-010
TRAINLM, Performance goal met.
```

网络经过 3 次训练后，目标误差达到要求，结果如图 10-12 所示。

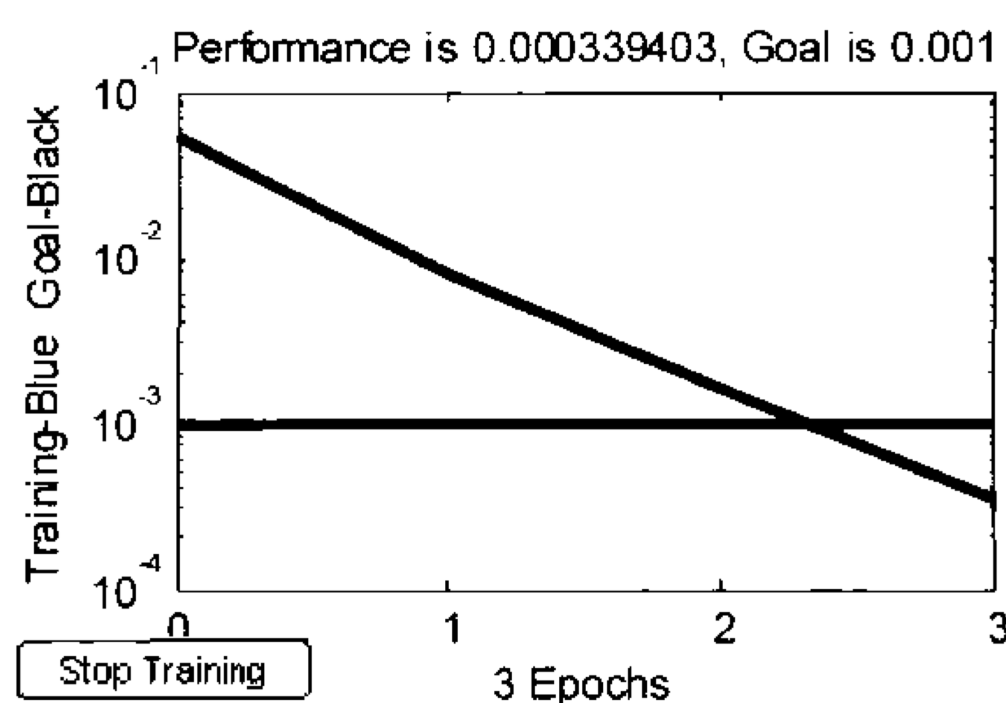


图 10-12　训练结果

预测过程代码为

```
P_test=[0.40    0.40 0.33 0.33 0.33 0.40 0.49 0.35;
        0.08    0.93 0.56 0.92 0.89 0.60 0.24 0.33;
        0.67    0.60 0.89 0.82 1.00 0.80 0.75 0.29;
        0.32    0.40 0.67 0.33 0.33 0.80 0.75 0.35;
        0.87    0.72 0.89 0.92 0.89 0.40 0.49 0.29]';
Y=sim(net,P_test)
```

输出结果为

```
Y =
    0.4641    0.1850    0.3676    0.5007    0.6614
```

从以上计算结果可以看出，方案 13 最优，方案 15 最差，这也与各方案的指标因素特点基本一致，根据此结果可进行配送中心选址的决策。

10.4　Elman 神经网络在信号检测中的应用

【例 10-2】利用 Elman 网络对信号幅度进行检测。

1）问题描述

待检测的信号是幅度变化的正弦信号，可以通过如下的语句生成。

```
P1=sin(1:20);
P2=sin(1:20) *2;
P=[P1 P2 P1 P2];
```

该信号由幅度分别为 1 和 2 的正弦信号交替变化构成，图 10-13 中所绘的虚线即是待测的正弦信号。本例中要利用 Elman 网络对信号的幅度进行检测，希望网络能够正确地输出时变信号的幅度值，因此网络的目标输出为

```
T1=ones(1,20);
T2=ones(1,20) *2;
T=[T1 T2 T1 T2];
```

目标输出曲线如图 10-13 中所绘的短画线。网络的输入和目标输出应采用串行序列格式。

```
Pseq=con2seq(P);
Tseq=con2seq(T);
```

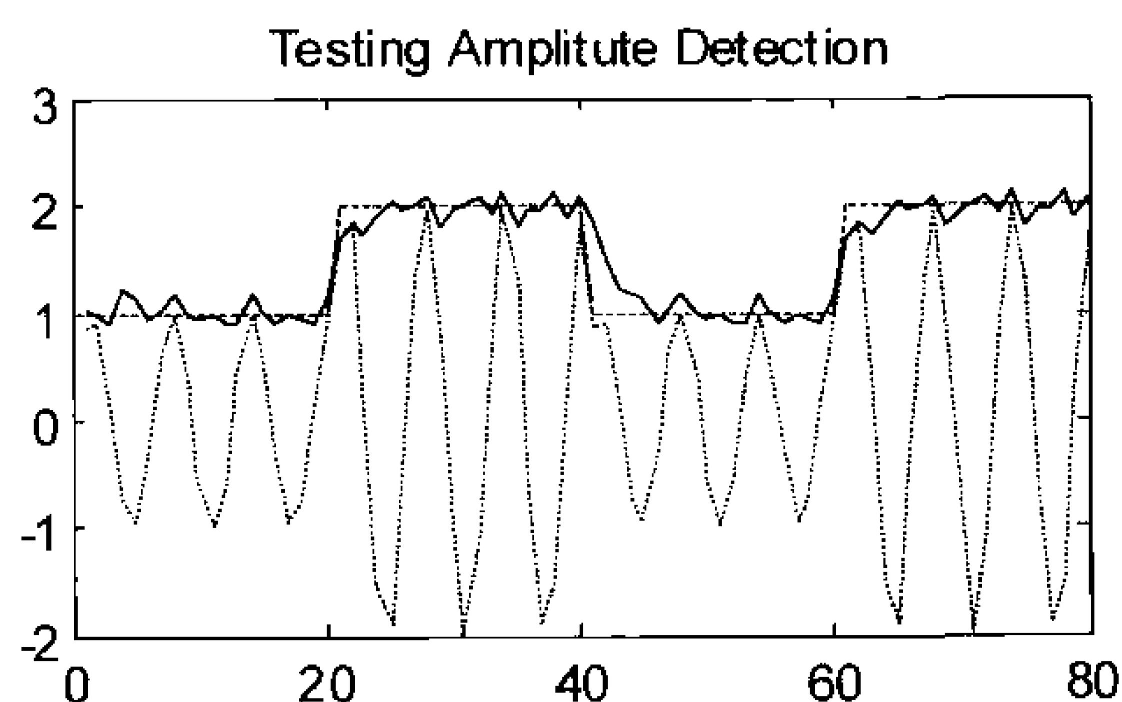


图 10-13　Elman 网络的信号幅度检测性能

2）网络设计

建立两层神经元构成的 Elman 网络，网络的隐层和输出层分别含有 10 个和 1 个神经元，传递函数分别为 tansig 函数和纯线性函数，网络训练函数为 traingdx 函数。

```
net=newelm([-2 2],[10 1],{'tansig','purelin'},'traingdx')
```

3）网络训练

设置好网络的训练参数后，就可以对网络进行训练了。

```
net.trainParam.epochs=1000;
net.trainParam.show=20;
```

```
net.trainParam.goal=0.01;
net.performFcn='sse';
[net,tr]=train(net,Pseq,Tseq);
```

训练结果如图 10-14 所示。

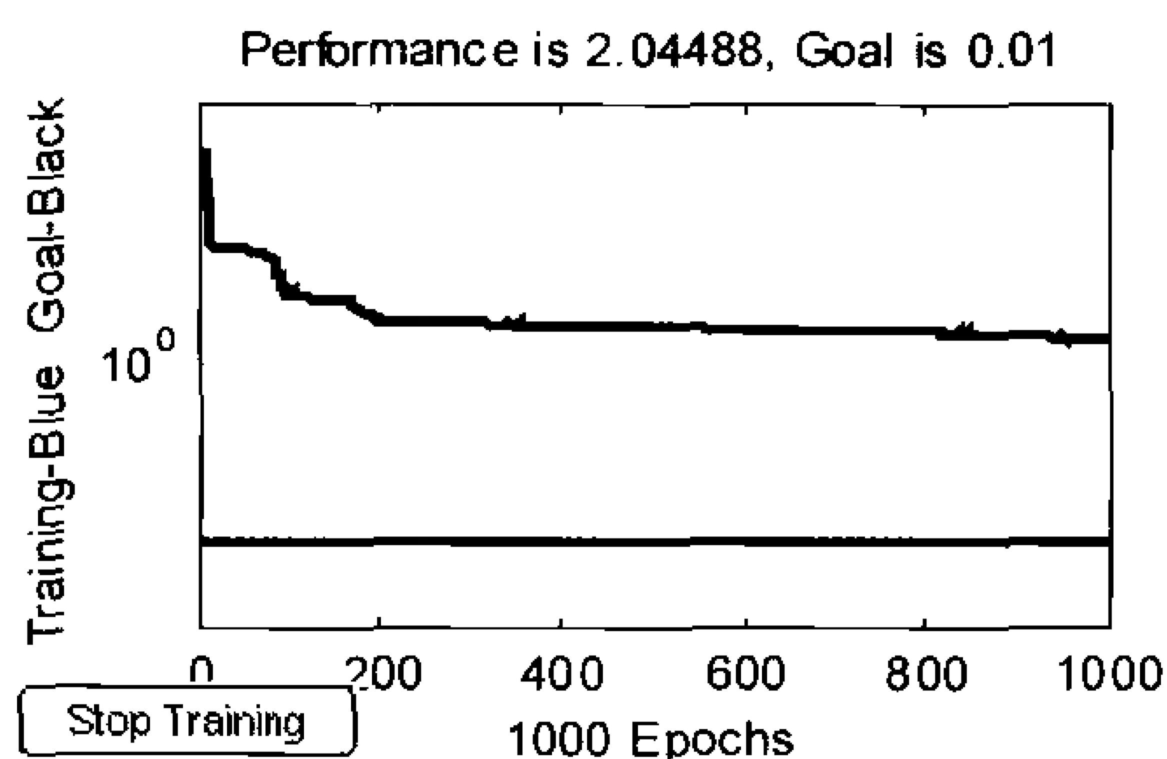


图 10-14　训练结果

网络训练结果为

```
TRAINGDX, Epoch 0/1000, SSE 260.584/0.01, Gradient 984.285/1e-006
TRAINGDX, Epoch 20/1000, SSE 23.7826/0.01, Gradient 17.9028/1e-006
……
TRAINGDX, Epoch 1000/1000, SSE 2.04488/0.01, Gradient 0.573174/1e-006
TRAINGDX, Maximum epoch reached, performance goal was not met.
```

4）对网络进行测试

利用样本输入数据对训练好的网络进行测试。

```
A=sim(net,Pseq);
```

并绘出样本输入、目标输出和网络仿真输出曲线。

```
time=1:length(P);
plot(time,P,':',time,T,'--',time,cat(2,A{:}));
title('Testing Amplitude Detection')
```

测试结果如图 10-13 所示，图中的虚线、短画线和实线分别表示网络输入的测试信号曲线、目标输出曲线和网络仿真输出曲线。从图 10-13 中可以看出网络对样本信号有较好的检测性能。

5）对网络推广性的测试

利用一组新的输入数据对训练好的网络进行测试。首先产生输入信号和目标输出。

```
P3=sin(1:20) *2.6;
T3=ones(1,20) *2.6;
P4=sin(1:20) *1.2;
T4=ones(1,20) *1.2;
Pg=[P3 P4 P3 P4];
Tg=[T3 T4 T3 T4];
Pgseq=con2seq(Pg);
```

然后利用这一输入信号对网络进行仿真。

```
A=sim(net,Pgseq);
```

并绘出测试信号、目标输出和网络仿真输出曲线。

```
figure;
```

```
time=1:length(Pg);
plot(time,Pg,':',time,Tg,'--',time,cat(2,A{:}));
title('Testing Generalization')
```

网络推广性能测试结果如图 10-15 所示，图中的虚线、短画线和实线分别表示网络输入的测试信号曲线、目标输出曲线和网络仿真输出曲线。从图 10-15 可以看出网络对这一测试信号的检测性能虽然不及样本信号，但依然可以较成功地实现幅度检测。

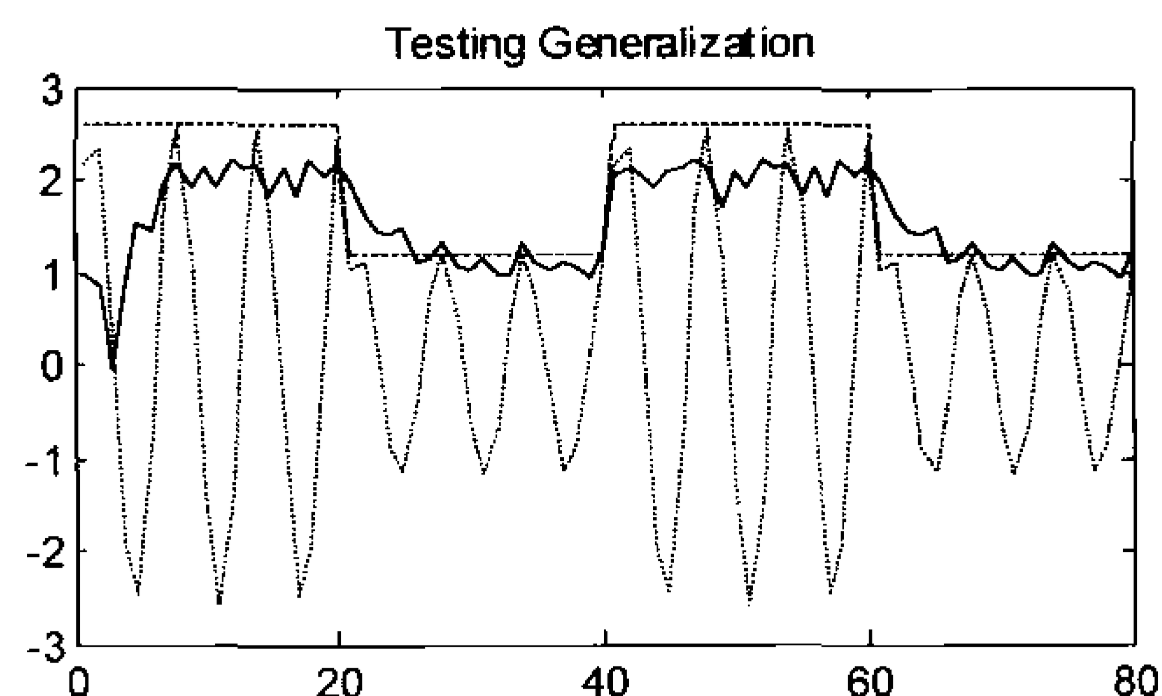


图 10-15　Elman 网络的推广性能测试结果

6）给出本例完整的 MATLAB 代码

```
%产生样本数据 P 和目标输出 T
P1=sin(1:20);
P2=sin(1:20) *2;
T1=ones(1,20);
T2=ones(1,20) *2;
%将上述两组数据合并得到网络的输入和目标输出
P=[P1 P2 P1 P2];
T=[T1 T2 T1 T2];
Pseq=con2seq(P);
Tseq=con2seq(T);
%建立 Elman 网络，网络由两层神经元构成
%两层分别有 10 个和 1 个神经元，传递函数分别为 tansig 函数和纯线性函数
%训练函数为 traingdx 函数
net=newelm([-2 2],[10 1],{'tansig','purelin'},'traingdx')
%对网络进行训练
net.trainParam.epochs=1000;
net.trainParam.show=20;
net.trainParam.goal=0.01;
net.performFcn='sse';
[net,tr]=train(net,Pseq,Tseq);
%利用样本数据对网络进行仿真
A=sim(net,Pseq);
time=1:length(P);
%画出样本数据的网络仿真输出图形
plot(time,P,':',time,T,'--',time,cat(2,A{:}));
title('Testing Amplitute Detection')
%利用一组新的数据对网络进行检测
P3=sin(1:20)*2.6;
```

```
T3=ones(1,20)*2.6;
P4=sin(1:20)*1.2;
T4=ones(1,20)*1.2;
%产生测试数据
Pg=[P3 P4 P3 P4];
Tg=[T3 T4 T3 T4];
Pgseq=con2seq(Pg);
%利用测试数据对网络进行仿真
A=sim(net,Pgseq);
%画出测试数据的网络仿真输出图形
figure;
time=1:length(Pg);
plot(time,Pg,':',time,Tg,'--',time,cat(2,A{:}));
title('Testing Generalization')
```

10.5 神经网络在噪声抵消系统中的应用

10.5.1 自适应噪声抵消原理

噪声消除是信号处理的核心问题之一，通常实现最优滤波的滤波器为维纳滤波器与卡尔曼滤波器。它们均要求已知信号和噪声的先验知识，但在许多实际应用中往往无法预先得知，为此发展了自适应滤波器。1965 年美国斯坦福大学建成了第一个自适应噪声抵消（ANC）系统。随着计算机技术与集成电路技术的进步，新的自适应算法不断涌现出来，自适应噪声抵消在理论和应用上都得到了很大发展。自适应噪声抵消技术是一种能够很好地消除背景噪声影响的信号处理技术。应用自适应噪声抵消技术，可在未知外界干扰源特征、传递途径不断变化，以及背景噪声和被测对象声波相似的情况下，有效地消除外界声源的干扰，获得高信噪比的对象信号。这一技术可为机械元件的噪声、振动等动态信号在测试环境不太理想的工作现场做测试分析和故障诊断时，提供有效的方法和依据，具有一定的理论意义和应用价值。

1．自适应滤波器

自适应滤波器自从 20 世纪 60 年代出现后，其理论在不断发展与完善，应用也越来越广泛。自适应数字滤波器的原理如图 10-16 所示。图中，$x(j)$：表示 j 时刻的输入信号值；$y(j)$：表示 j 时刻的输出信号值；$d(j)$：表示 j 时刻的参考信号值或期望响应的信号值；$e(j)$：误差信号，$e(j)=d(j)-y(j)$。

自适应数字滤波器的滤波参数受误差信号 $e(j)$ 的控制，根据 $e(j)$ 的值而自动调整，以便使输出 $y(j+1)$ 接近于所期望的参考信号 $d(j+1)$。

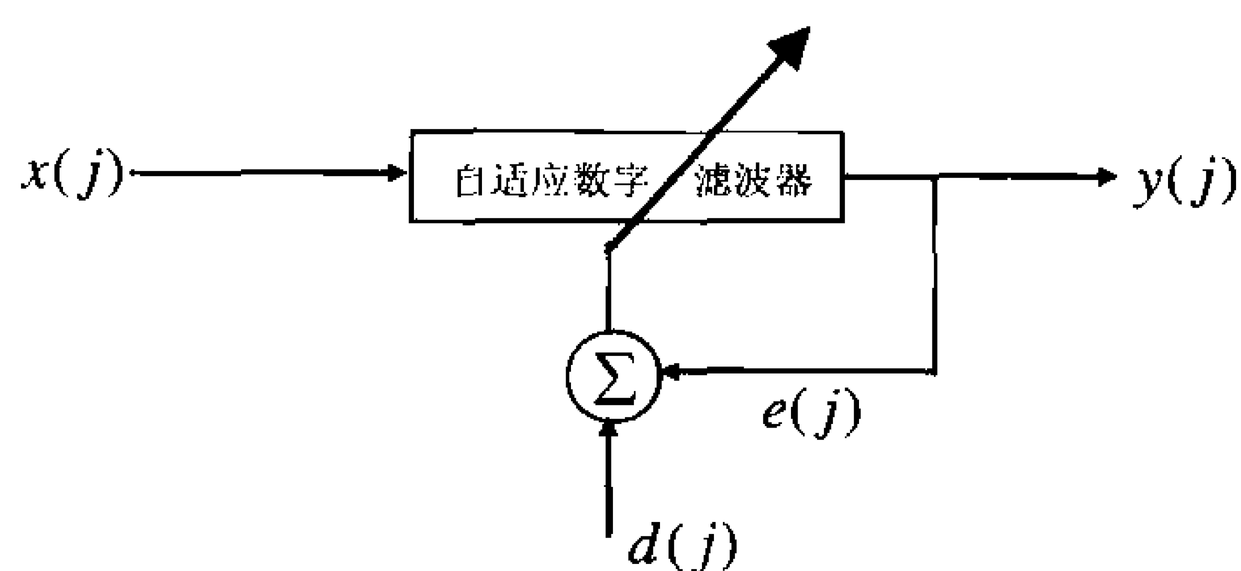


图 10-16　自适应数字滤波器原理

2. 自适应噪声抵消系统基本原理

自适应噪声抵消系统除了需要原始输入外，还需要一个参考输入，供给与原始输入相关的噪声，以便抵消原始输入中的噪声，而其中的有用信号几乎不产生什么畸变。

图 10-17 描述的是一典型的自适应噪声抵消系统，其中原始输入信号 $d(k)$ 是有用信号 $s(k)$ 与噪声 $z(k)$ 之和，参考输入信号 $x(k)$ 是与 $z(k)$ 相关的噪声 $c(k)$ 。假设 $s(k)$ 、 $z(k)$ 与 $c(k)$ 是零均值的平稳随机过程，$s(k)$ 与 $z(k)$ 、$c(k)$ 不相关。由图 10-17 可见，自适应滤波器的输出 $z_o(k)$ 为 $c(k)$ 的滤波信号，因此，自适应噪声抵消系统的输出 $y(k)$ 为

$$y(k)=s(k)+z(k)-z_o(k)$$

而

$$y^2(k)=s^2(k)+[z(k)-z_o(k)]^2+2s(k)[z(k)-z_o(k)]$$

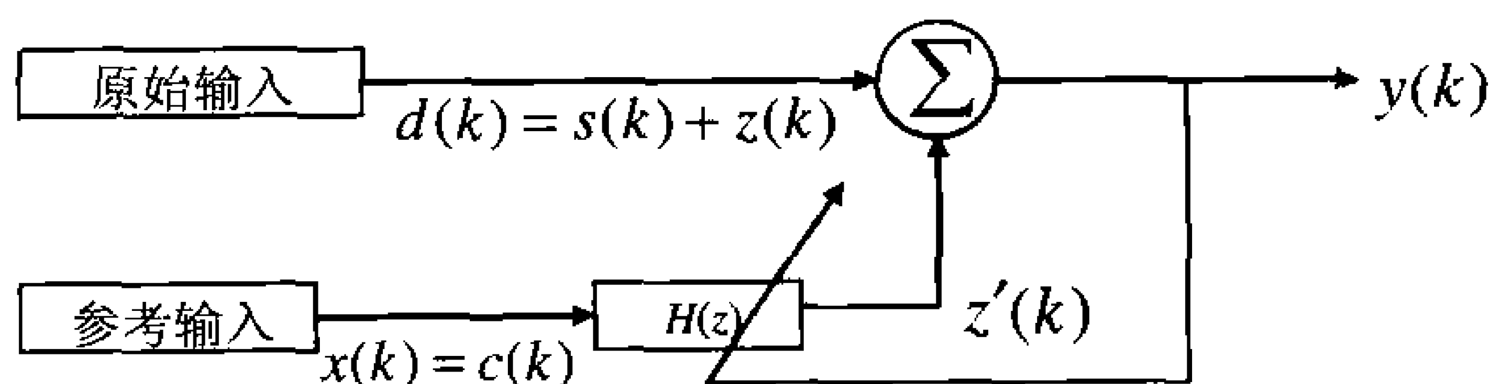


图 10-17　自适应噪声抵消系统

根据前面不相关的假定，对上式两边取数学期望，可得

$$\mathrm{E}[y^2(k)]=\mathrm{E}[s^2(k)]+\mathrm{E}[z(k)-z_o(k)]^2$$

信号功率 $\mathrm{E}[s^2(k)]$ 与自适应滤波器的调节无关，因此，自适应滤波器调节使 $\mathrm{E}[y^2(k)]$ 最小，也就是 $\mathrm{E}[z(k)-z_o(k)]^2$ 最小， $\mathrm{E}[(y(k)-s^2(k))^2]$ 也最小，即自适应噪声抵消系统的输出信号 $y(k)$ 与有用信号 $s(k)$ 的均方差最小。

在理想情况下， $z_o(k)=z(k)$ ，则 $y(k)=s(k)$ ，这时，自适应滤波器自动调节其脉冲响应，将 $c(k)$ 加工成 $z(k)$ ，与原始输入信号 $d(k)$ 中的 $z(k)$ 相减，使输出信号 $y(k)$ 由于噪声完全被抵消而等于有用信号 $s(k)$ 。

可以证明，自适应滤波器能完成上述任务的必要条件为：参考输入信号 $x(k)=c(k)$ 必须与被抵消的信号（现为噪声） $z(k)$ 相关。

一个实际的噪声抵消系统的情况比如图 10-17 所示的还要复杂一些，这是因为输入的还可能是一些独立的噪声源，即与参考输入无关的噪声及干扰，如图 10-18 所示。

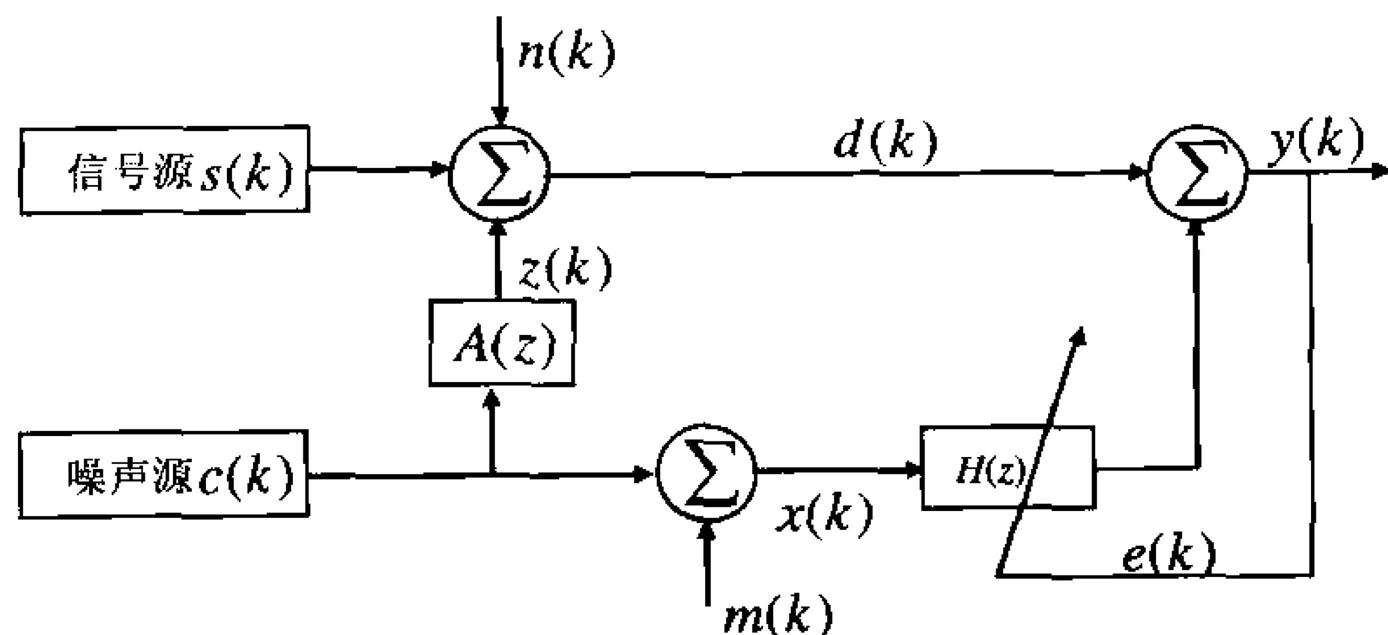


图 10-18　常见的自适应噪声抵消系统

10.5.2 噪声抵消系统的 MATLAB 仿真

1. BP 网络模型建立

根据分析结论，这里构造一个 1-4-1 型 BP 神经网络，隐含层节点有 4 个，如图 10-19 所示。

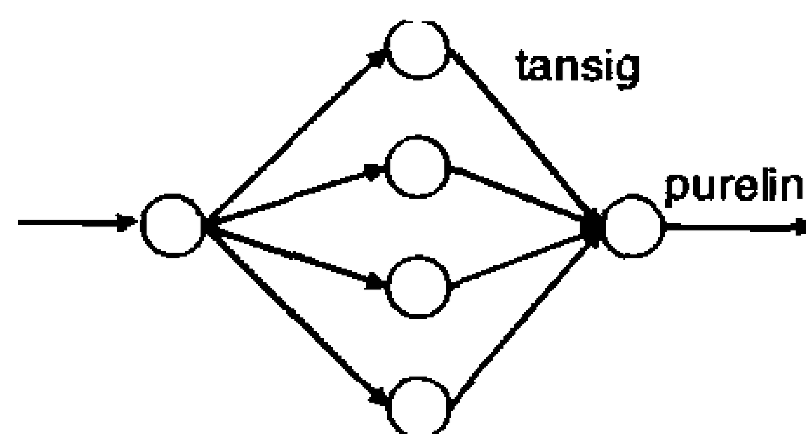


图 10-19　BP 神经网络模型

本网络中的隐含层变换函数为 tansig，是可微函数，它可以将神经元的输入范围(−∞,+∞)映射到（−1,+1），非常适合于训练 BP 网络的神经元。如果 BP 网络的最后一层是 Sigmoid 型神经元，那么整个网络的输出就会限制在一个较小的范围内；如果是 purelin 型线性神经元，则整个网络的输出可为任意值。

2. 基于神经网络工具箱的 BP 网络学习和训练

神经网络工具箱为训练神经网络提供了帮助，可以利用它提供的函数对网络进行初始化、仿真和训练，并通过变化的图形观察其动态训练过程。

如图 10-20 所示，给出了 BP 神经网络的训练过程。

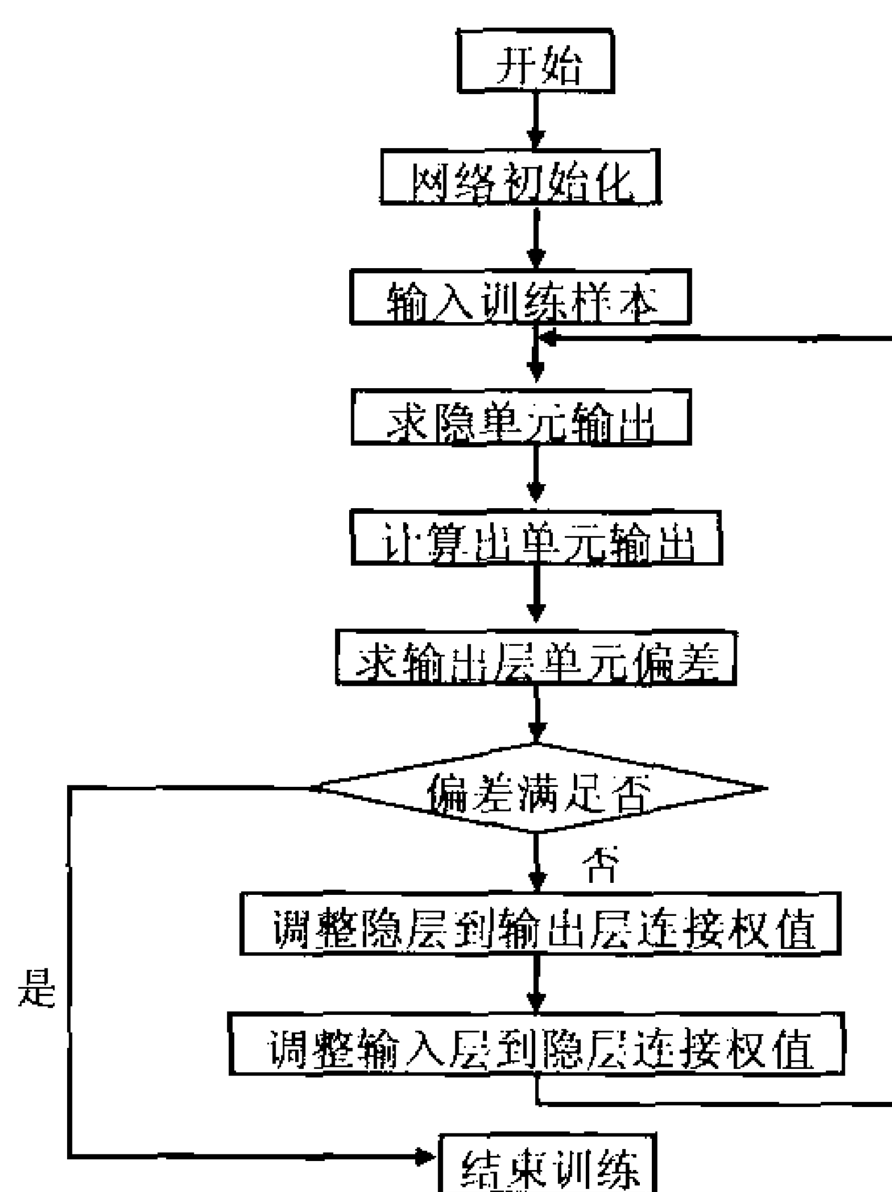


图 10-20　BP 神经网络训练过程

以下是 MATLAB 的实现代码。

```
%定义样本
T=0:0.01:10;
Y=randn(size(T));
Yn=sin(Y);
P=Y(1:30)
T=Yn(1:30)
%初始化 BP 网络
%其中，P 为输入量，隐层节点数为 4，输出层节点数为 1
%tansig 和 purelin 分别为隐层和输出层的变换函数
```

```
%训练算法为 trainlm
net=newff(minmax(P),[4,1],{'tansig','purelin'},'trainlm');
%BP 网络训练
%利用 BP 学习规则训练前向网络，使其完成函数逼近、矢量分类和模式识别，选择训练参数，
%并指示如何进行训练
%一旦训练达到最大的训练次数或网络误差平方和降低到误差之下，都会使网络停止学习
%指定两次更新显示间的训练次数
net.trainParam.show=10;
%指定训练的最大次数
net.trainParam.epochs=1000;
%误差平方和指标
net.trainParam.goal=0.001;
%指定学习速率，即权值和阈值更新的比例
net.trainParam.lr=0.01;
%开始训练
[net,tr]=train(net,P,T);
%画出误差变化曲线
figure(1);
plotperf(tr);
%计算网络仿真输出
A=sim(net,P);
figure(2);
plot([0:0.1:2.9],T,'b+-',[0:0.1:2.9],A,'r+*');
```

在这段程序中，BP 网络训练时，函数利用单层权值矢量 ***W***、阈值矢量 ***B*** 及转移函数成批训练网络，使当输入 ***P*** 时，网络的输出为目标矢量 ***T***。在这个环节将得到新的权值矢量 ***W*** 和阈值矢量 ***B***，以及记录网络训练过程的误差平方和行向量 tr。

误差训练结果为

```
TRAINLM, Epoch 0/1000, MSE 4.63725/0.001, Gradient 117.172/1e-010
TRAINLM, Epoch 3/1000, MSE 3.87303e-005/0.001, Gradient 0.0401599/1e-010
TRAINLM, Performance goal met.
```

误差变化曲线如图 10-21 所示，函数逼近曲线如图 10-22 所示。

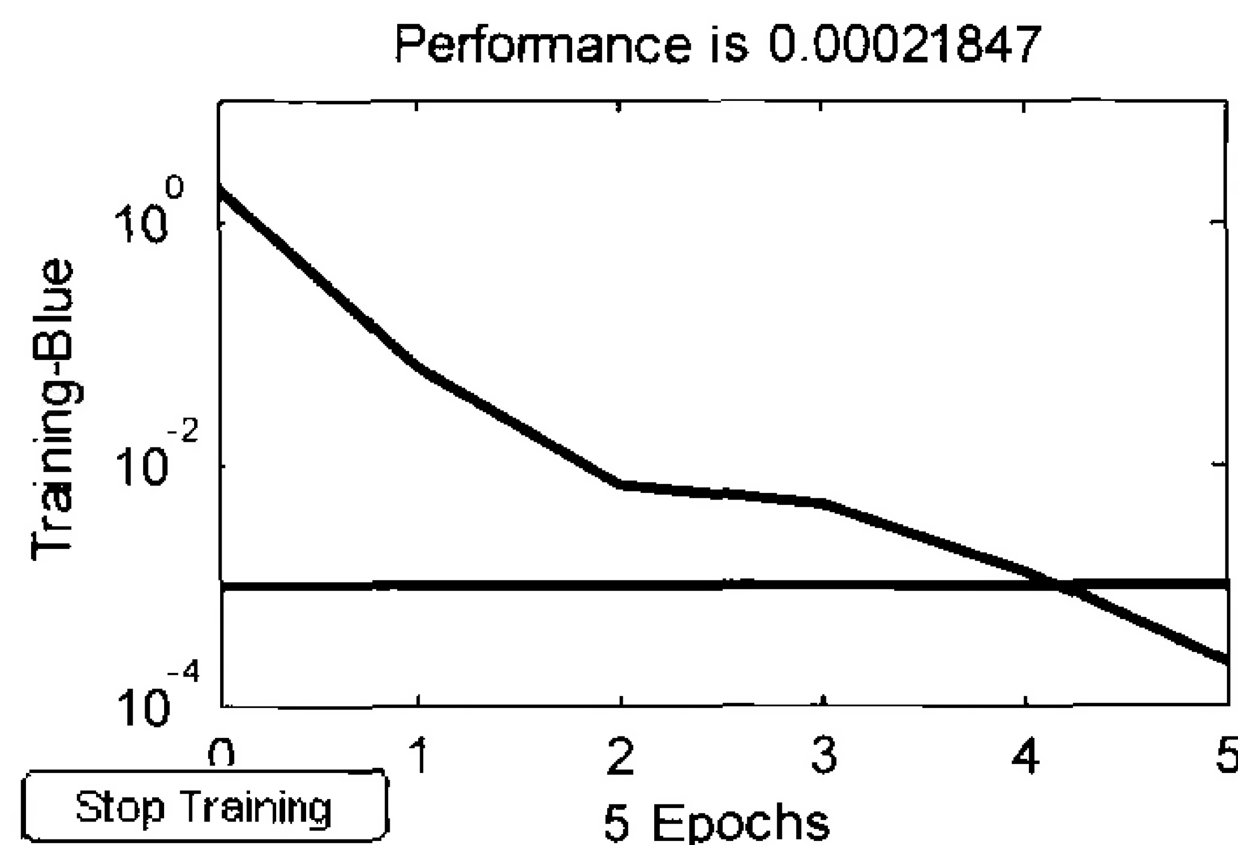


图 10-21　误差变化曲线

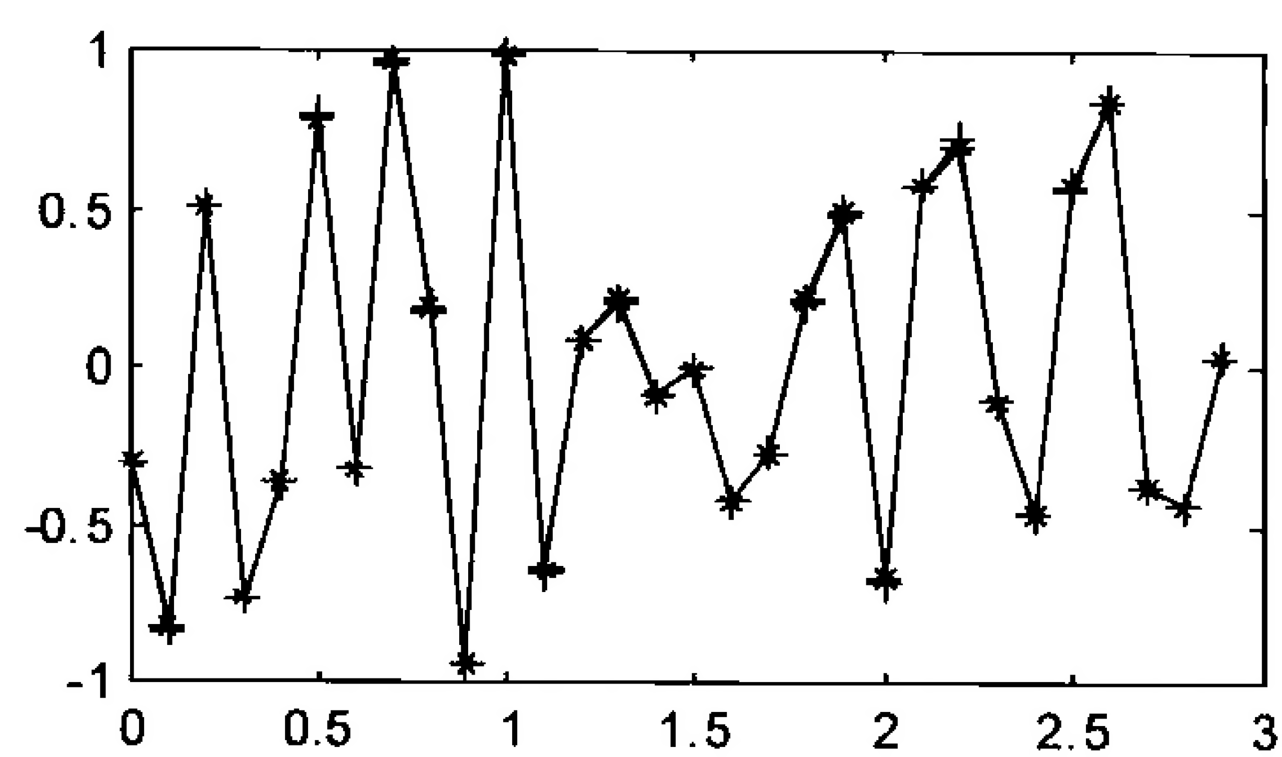


图 10-22 函数逼近曲线

根据误差曲线的变化，可以看到经过 600 多次训练之后，误差最终逼近于 0.01。从函数逼近曲线可看出输出曲线非常逼近于目标曲线，证明训练后的网络具有较好的拟合性，本环节训练了网络并得到了构造 BP 网络的基本要素——权值和阈值。

参 考 文 献

[1] 飞思科技产品研发中心. 神经网络理论与 MATLAB 实现. 北京：电子工业出版社，2005

[2] 飞思科技产品研发中心. MATLAB 6.5 辅助图像处理. 北京：电子工业出版社，2003

[3] 田景文，高美娟. 人工神经网络算法研究及应用. 北京：北京理工大学出版社，2006

[4] Simon Haykin. 神经网络的综合基础（第 2 版）. 北京：清华大学出版社，2001

[5] 王士同. 神经模糊系统及其应用. 北京：北京航空航天大学出版社，1998

[6] 张智星，孙春在，等. 神经-模糊和软计算. 西安：西安交通大学出版社，2000

[7] 张瑞丰. 精通 MATLAB 6.5. 北京：中国水利水电出版社，2004

[8] 高隽. 人工神经网络原理及仿真实例（第 2 版）. 北京：机械工业出版社，2007

[9] 杨行峻，郑君里. 人工神经网络. 北京：高等教育出版社，1992

[10] 沈清，胡德文，时春. 神经网络应用技术. 长沙：国防科技大学出版社，1993

[11] 张立明. 人工神经网络模型及其应用. 上海：复旦大学出版社，1993

[12] 张铃，张钹. 人工神经网络理论及应用. 杭州：浙江科学技术出版社，1997

[13] 张良均，曹晶，蒋世忠. 神经网络的实用教程. 北京：机械工业出版社，2007

[14] 弱兆礼，赵春晖，梅晓丹. 现在图像处理技术及 MATLAB 实现[M]. 北京：人民邮电出版社，2001

[15] 马兴义，杨立群，林敏，龚少华. MATLAB 6.5 应用接口编程. 北京：机械工业出版社，2002

[16] 何强，何英. MATLAB 扩展编程. 北京：清华大学出版社，2002

[17] 闻新，周露，李翔，等. MATLAB 神经网络仿真与应用. 北京：北京科学出版社，2003

[18] 韩力群. 人工神经网络理论设计与应用. 北京：化学工业出版社，2002

[19] 阎平凡，张长水. 人工神经网络与模拟进货计算（第 2 版）. 北京：清华大学出版社，2005

[20] 焦李成. 神经网络的应用与实现. 西安：西安电子工业出版社，1996

[21] 葛哲学. MATLAB R2007 基础与提高. 北京：电子工业出版社，2007

[22] 焦李成. 神经网络系统理论. 西安：西安电子科技大学出版社，1990

[23] 施晓红，周佳. 精通 GUI 图形界面编程. 北京：北京大学出版社，2003

[24] 王正林，刘明. 精通 MATLAB 7. 北京：电子工业出版社，2006

[25] Fredric M.Ham，等. 神经计算原理. 北京：机械工业出版社，2003

[26] 李显宏. MATLAB 7.X 界面设计与编译技巧. 北京：电子工业出版社，2006

[27] 赵震宇，徐用懋. 模块理论和神经网络的基础和应用. 北京：清华大学出版社，1996

[28] 龙脉工作室，刘会灯，朱飞. MATLAB 编程基础与典型应用. 北京：人民邮电出版社，2008

[29] 罗四维. 人工神经网络建造. 北京：中国铁道出版社，1998

[30] 杨铭震，王燕霞. 人工神经网络及其在石油勘探中的应用. 北京：兵器工业出版社，1993

[31] 李弼强，彭天强，彭波，等. 智能图像处理技术. 北京：电子工业出版社，2004

[32] 葛哲学. 精通 MATLAB. 北京：电子工业出版社，2008

[33] 边肇祺，张学工. 模式识别（第二版）. 北京：清华大学出版社，2000

[34] 王正林，龚纯，何倩. 精通 MATLAB 科学计算. 北京：电子工业出版社，2007
[35] 殷勤业，等. 模式识别应用. 北京：机械工业出版社，1999
[36] 董振海. 精通 MATLAB 7 编程与数据库应用. 北京：电子工业出版社，2007
[37] 刘卫国. MATLAB 程序设计与应用（第二版）. 北京：高等教育出版社，2006
[38] 何友，王国宏. 多传感器信息融合及应用. 北京：国防工业出版社，2005
[39] 魏海坤. 神经网络结构设计的原理与方法. 北京：中国铁道出版社，2000
[40] 李士勇. 模糊控制、神经控制和智能控制论. 哈尔滨：哈尔滨工业大学出版社，1996